THE UNIVERSITY *of* EDINBURGH

A COMPUTER MODEL OF CONVERSATION

by

RICHARD POWER

Ph.D. University of Edinburgh 1974

Abstract

This paper is addressed to the problem of how it is possible to conduct coherent, purposeful conversations. It describes a computer model of a conversation between two robots, each robot being represented by a section of program. The conversation is conducted in a small subset of English, and is a mixed-iniative dialogue which can involve interruptions and the nesting of one segment of dialogue in another.

The conversation is meant to arise naturally from a well-defined setting, so that it is clear whether or not the robots are saying appropriate things. They are placed in a simple world of a few objects, and co-operate in order to achieve a practical goal in this world. Their conversation arises out of this common aim; they have to agree on a plan, exchange information, discuss the consequences of their actions, and so on.

In previous language-using programs, the conversation has been conducted by a robot and a human operator, rather than by two robots. In these systems, it is almost always the human operator who takes the initiative and determines the overall structure of the dialogue, and the processes by which he does so are hidden away in his mind. The aim of our program is to make these processes totally explicit, and it is for this reason that we have used two robots and avoided human participation. Thus the main focus of interest

is not the structuring of individual utterances, but the higher-level organisation of the dialogue, and how the dialogue is related to the private thoughts which underlie it.

The program has two kinds of procedure, which we call ROUTINES and GAMES, the Games being used to conduct sections of conversation and the Routines to conduct the underlying thoughts. These procedures can call each other in the normal way. Thus the occurrence of a section of dialogue will be caused by the call of a Game by a Routine; and when the section of dialogue ends, the Game will exit, returning control to the Routine which called it.

There are several Games, each corresponding to a common conversational pattern, such as a question and its answer, or a plan suggestion and the response to it. The Games determine what can be said, who will say it, how each remark will be analysed, and how it will be responded to. They are thus joint procedures, in which the instructions are divided up between the robots. When a section of dialogue occurs, the relevant Game will be loaded in the minds of both robots, but they will have adopted different roles in the Game, and will consequently perform different instructions and make different utterances.

Contents

Preface

The work reported in this paper was carried out between October 1971 and March 1974 at the Theoretical Psychology Unit of the School of Artificial Intelligence at Edinburgh University. It was financed by grants from the Science Research Council and the Royal Society. I would like to thank all the members of the Theoretical Psychology Unit for their help, and in particular Professor Christopher Longuet-Higgins, who supervised the project.

1. INTRODUCTION

1.1 The Problem

This paper describes a model in which two robots hold a conversation, the robots being sections of computer program and the conversation being the output of the program. The robots can carry out a variety of conversations, depending on their prior goals and beliefs, and they carry out these conversations without any assistance from a human operator. They inhabit a simple world of a few objects, and can be given practical goals to achieve in this world; the dialogue arises when they co-operate to achieve an agreed goal. It is a mixed-iniative dialogue conducted in a small subset of English, and the program is written so that a robot can also co-operate with a human operator, provided that the operator stays within the subset of English and sticks to certain conversational forms.

A conversation is not just a random sequence of sentences, even if all the sentences are grammatical and meaningful. To be acceptable, a conversation must consist of remarks which are appropriate, both to each other and to a readily imaginable context. As an example of an inappropriate conversation, consider the following joke:

(1) The scene is a bus.

(a) Passenger: What's the time?

(b) Conductor: Thursday.

(c) Passenger: In that case I'd better get off.

The inappropriate utterances here are (1b) and (1c); (1b) is only sensible as an answer to a question such as "What day is it?", and (1c) should follow a statement such as "We're at Marble Arch" since one gets off buses at the right place, not at the right time. But although (1b) and (1c) are inappropriate, they should be distinguished from utterances such as:

(2) Colourless green ideas sleep furiously,

the famous sentence invented by Chomsky as an example of grammatical nonsense. The difference between (1c) and (2) is that while (1c) is nonsense in its particular context, (2) would be nonsense in any context, since it is deliberately crammed with conceptual contradictions. This distinction might be made by saying that (1c) is meaningful but inappropriate, while (2) is meaningless, and therefore barred from the possibility of being appropriate.

In order to be appropriate, then, an utterance must be more than meaningful. Specifically, it must meet two further conditions: first, it must be relevant to the underlying situation, and second, it must fit into the previous conversation. Examples (3) and (4) bring out this distinction.

(3) First man: Look - a tiger!

Second man: Poor beast looks underfed.

(4) First man: What time is it?

Second man: I agree.

In example (3), we imagine that the two men are walking

along an ordinary street when they meet a tiger which has escaped from the local zoo. We assume that neither man relishes the thought of being eaten, and thus that their main goal will be to get out of the tiger's way as speedily as possible. In this context, the first man's remark is sensible enough, but the second man's comment is totally irrelevant. Had the tiger been locked up in a zoo, however, the comment would have been a perfectly reasonable expression of concern for the animal's welfare. It was only the context which made it inappropriate, the context being defined by the situation of the speaker and his goals.

In example (4) the second man again says something inappropriate, but this time there is no other situation in which the remark might be a reasonable response. The first man asked a question, and if you are asked a question you are expected to either give a sensible answer or give a reason for not answering, e.g. "Ten to five" or "My watch has stopped". The second man does neither; instead, he replies as if the first man had made a statement or suggested a plan. If a conversation is to make coherent sense, the parties must do more than reflect their individual goals; they must also respond to what has just been said: answer questions, reply to suggestions, comment on statements, and so on.

If we are to understand a dialogue, then, we must take three factors into account: first, the goals of speaker A; second, the goals of speaker B; and third, the conversational

conventions shared by A and B. Let us take a typical short dialogue, example (5), and analyse it in terms of these factors.

(5) (a) Boy: Shall we go to a movie tonight?

(b) Girl: What's on?

(c) Boy: A Charlie Chaplin film.

(d) Girl: Was he in "Modern Times"?

(e) Boy: Yes.

(f) Girl: All right, I'll come.

This dialogue divides up most naturally into three pairs of utterances, (a + f), (b + c), and (d + e). In each pair, the second utterance is a reply to the first: either a response to a suggestion, or an answer to a question. We also note that the pairs (b + c), (d + e) are nested inside the outer pair (a + f), and are to some extent related to it. The boy's aim in utterance (a) is obviously to go to the movie with the girl: this goal will probably be related to a number of higher goals, but we can ignore these goals for present purposes. The girl's motives in (b) and (d) are less clear: she may want to make sure the movie is worth seeing before she agrees, or she may have already decided to go and just be asking out of interest. The boy is probably not sure what her motives are at (b) and (d), but he answers the questions all the same, since to do so is a virtually automatic response unless one is especially on one's guard.

We have given an informal account of the causes of the utterances in (5); the other side of the coin is to give an account of their effects. Here the division into three pairs is especially apposite, since each pair achieves a separate result. The pair (a + f) gives rise to an agreed course of action, and the pairs (b + c) and (d + e) cause information to be conveyed to the girl's memory. Thus if we want to represent the causes and effects of conversations, we will need to have a way of describing states of mind; more specifically, we will have to represent the goals, plans, and memories of the two speakers.

The conversation analysed above is an exceptionally simple one, in which the goals of the speakers are reasonably intelligible and the result of the dialogue is not hard to describe informally. Most conversations are psychologically far more intricate: when two people have a chat to get to know one another, or have a discussion or argument, it is not so easy to say why a given utterance was made or what it achieved. The simplest conversations arise when the parties are co-operating to achieve a concrete objective. Then the goals are ordinary practical ones rather than complex psychological ones, and it is much easier to trace their effect on the dialogue. If, for example, the objective is to open a locked door, then it is clear that remarks such as "Let's search the drawer for the key", or

"Where are the keys?" are relevant, because they fit in with the natural division of the task into sub-goals. In our program we have modelled a situation of this kind, so that the goals of the participants are entirely practical ones. There are two robots put in a simple world, and they share the goal of bringing about a particular change in that world.

This being so, it is necessary for the robots to be able to act and plan as well as speak. If they are going to co-operate, presumably they must both carry out actions, and these actions must be performed in the right order. So there must be a master plan which they are following, and each robot must know the plan. Moreover, the plan has to be thought up in the first place, and must be modified if it goes wrong. To get from a state in which both parties are thinking up plans to a state in which a plan is agreed can be a complicated process. Suppose, for instance, that before the conversation in example (5) the boy and girl had agreed to go out and were wondering where to go. Presumably they are both mulling over various possibilities in their heads: the boy is thinking about the movie, and the girl would like to go for a drink and is wondering which pub to suggest. Eventually the boy decides to speak, finds a suitable remark, makes it, and awaits the response. The girl now has to break off her mental pub crawl and address

herself to his suggestion: does she want to go to the movie? If she decides to say no, she must return to her earlier train of thought about pubs, and come up with the suggestion she was just about to make. If she says yes, on the other hand, she must leave off the previous train of thought and avoid making her suggestion. In the example, she does something even more complex: she starts a new train of thought about films, and decides to find out more about the film before responding; having asked the questions (b) and (d) she breaks off this new line of thought, and returns to the task of replying to (a).

These operations are ones which we all carry out with great ease, and we regard them as simple common sense. It is only when we try to state the operations formally that we realise there are problems here at all, or see exactly what the problems are. In the list which follows, we draw attention to some of the abilities which human speakers have and which a good conversational program would also need. It helps to have an example before us, so we will imagine that there are two participants, A and B, and that A asks B a question. Although the list below is mainly built around this example, it is intended to illustrate general features which apply to all conversational interactions.

1. Presumably A has been trying to solve some problem, and has got stuck through ignorance of a particular

item of knowledge, which we will call K. How does he know what it is that he needs to find out?

2. How does A know that asking the question might lead to his knowing K?
3. Why does he expect B to reply?
4. How does A know how to interpret B's reply?
5. How does A know what range of replies to his question is appropriate?
6. How does A remember that he has asked the question, so that he doesn't ask it twice?
7. How does B know he is meant to reply?
8. How does B know which range of replies is appropriate?
9. How is B able to reply in so many different ways: e.g. by simply answering, or refusing to answer, or saying he doesn't know, or denying a pre-supposition of the question, or disputing the sincerity of the question, or asking what one of the words means, or ignoring it altogether? How is a question able to direct his mind to any of these various responses?
10. Why do we normally answer questions so automatically? What extra processes are involved when we are on our guard?
11. When A asks his question, how can B break off his train of thought and switch to a topic which may be totally different, and wholly unconnected to his goals?

12. If A asks a difficult question which requires thought, how can B turn his mind to solving the necessary, problems, and then remember that he has to give a reply?

13. If B was just about to tell A the knowledge K when A asked the question, how does he know there is no need to tell A anything once he has given the answer? In other words, how does B realise that his earlier intention is no longer relevant, and cancel it?

14. How can B tell whether or not he understands A's question?

15. And how is A able to replace his question by a paraphrase if B fails to understand it?

16. After interpreting B's reply to his question, how is A able to return to his previous train of thought at the right place?

17. And how can B return to his train of thought even though the conversation was in no way related to it?

18. If B says he doesn't know the answer, how does A avoid repeating his question when he returns to his previous thoughts? And how does he avoid asking the question if it crops up in a different context?

19. If B replies by asking a different question, how does A know that B is not yet answering him?

20. And if A answers B's question, and then B answers his, how does A remember what he originally asked, and not

muddle it up with the intervening dialogue? In short: how do we conduct nested conversations?

21. Why do people find it impossible to nest conversations too deeply?

22. When people return to a topic after a diversion, why do they like to reiterate the position in general terms?

23. And how does this process occur; e.g. what is the effect on the hearer's mind of a remark such as "Let's finish deciding who to invite to dinner"?

24. How do we construct and store a general description of an earlier conversation? What does the description look like?

25. And when we return to a conversation, how do we link a new, specific dialogue on to the general description? In other words, how is the general description used to help choose our further remarks?

We will regard these 25 problems as the main observations which a theory of appropriate conversation has to account for. The list is doubtless incomplete: for example, it entirely ignores the range of problems associated with learning and development. What is more, the problems have only been formulated in a primitive way, and they are not at all well-defined. But it helps to start off with some idea of what we want to explain, and we will use the list

above for this purpose. The general problem, of which the list is a more detailed statement, might be summarised as follows: we want to explain how people organise conversations, and how these conversations are related to their other thoughts.

1.2 Methodology

There are three methodological problems which arise during an investigation into conversation: first, what kind of data should be considered; second, what kind of theory should be sought; and third, how should different theories be evaluated. In this section we examine these issues in turn.

Data

Given two speakers and some context, there will be wide agreement among observers as to whether or not the ensuing conversation is appropriate, just as there is wide agreement on whether a given sentence is grammatical. We can all generate typical conversations with ease; novelists and playwrights make their living by doing so. But the ease with which we produce and recognise appropriate conversations can be misleading, since it may lead us to believe that the explanation of these abilities is a simple matter. In fact, if one considers the set of possible conversations, it is evident that only a tiny subset of these possibilities would be judged appropriate. If one took a piece of

dialogue from a play, and shuffled the utterances around, the result would almost certainly be dismissed as unacceptable by all observers. Since the task of producing acceptable conversations is thus non-trivial, and since at this stage there is no reasonable theory of how we do it, it is sensible to rely on our everyday knowledge as a source of data: in other words, to imagine some simple conversations which would be generally understood and accepted, and to seek to give an account of how these conversations could have come to pass.

The linguistic evidence to be explained is thus in a sense common knowledge. But it is important to note that this is only true at a certain level of description, the level based on concepts such as "word" and "sentence". The linguistic stimulus could also be described in terms of the mouth and throat movements which occurred during its production, or in terms of the frequency pattern of its sounds. At the other end of the scale it could be described according to its meaning, if we had a perfectly unambiguous artificial language suitable for framing such a description. None of these descriptions arises in everyday experience: people are usually unconscious of their vocal movements, and an adequate method of formalising meaning has not yet been discovered. In accepting the observations of everyday life, we are assuming that the description in terms of

words and sentences is psychologically important, and that it can form the basis for the higher-level descriptions that we want to make. This seems a reasonable assumption, but it is worth remembering that there are some important features - intonation and stress for example - which are lost when we rely on a list of words as our basic repres-entation of the evidence.

Data is seldom "raw"; in collecting and organising it one inevitably makes decisions as to how it should be described, and these decisions can only be made sensibly in the light of a pre-existing theory, even if the theory is only a vague one. It would be possible to assemble massive records of actual dialogues, and to carry out a statistical analysis to find out the frequencies of occurrence of various types of utterance, or the probability of one utterance given another, and so on; but there is no reason to believe that anything useful would emerge from this labour: the statistical description misses the point. The list of observations given at the end of 1.1 is quite different in character, and it is no accident that the list was compiled after writing the program and not before, and that the observations are expressed in highly abstract terms. It is not at all obvious which abstract categories are apposite for this task, and it is for this reason that we have made no effort to collect reams of data and to

organise it in terms of categories which would almost certainly turn out to be irrelevant.

Theory

The greatest recent influence on language theorising has been that of Chomsky (1957, 1965). Chomsky's main interest is different from that of the present work - he is concerned with how people put together words to make sentences, while we are concerned with how they put together sentences to make conversations - but it is useful nonetheless to examine his way of approaching the subject. In brief, Chomsky's position is this: native speakers of a language agree widely as to which arrangements of words are grammatical and which are not, and it is reasonable to suppose that the knowledge which they use in carrying out this task is common to a large degree. It is possible to distinguish the knowledge a person has of his language - his competence - from the processes which occur when he produces or perceives particular utterances, these being matters of performance. When a person perceives an utterance, for example, he uses not only his knowledge of the language, but also his knowledge of the context, the speaker, and so on, and thus competence and performance are separate though related issues. Chomsky's aim is to provide a formal description of linguistic competence: that is, he wants to describe the linguistic knowledge of the language-user in a totally explicit way.

It is difficult to see how one could have a competence model of conversation corresponding to Chomsky's theory of grammar. Whether or not a sentence is grammatical is a purely linguistic matter, and sentences are far more regular and unitary than conversations. In understanding conversations it is essential to consider the non-linguistic context and the goals of the speakers, and since the conversation is organised from two sources rather than one, it cannot usually be described as an orderly hierarchy leading up to a single node. The interesting issues in conversation are to do with performance: how language is used to further the goals of the speakers; how mixed-initiative dialogues are conducted; and so on. For these reasons, we have modelled performance, not competence. In abandoning competence, however, we do not abandon explicitness: just as Chomsky aims to describe linguistic knowledge by formal rules of grammar, so we aim to describe conversations by tracing the exact structures and processes which are brought into play when the model produces them.

When Chomsky puts forward a grammar as a theory of competence, he makes no effort to apply the grammar to hundreds of actual cases; instead, he gives only a few rules together with one or two examples to show how they work. The reason for this is that he is more interested in finding what <u>kind</u> of rule is required than in applying

a particular kind of rule exhaustively. Thus in "Syntactic Structures" (1957) he mentions three kinds of grammar: finite state grammar, phrase structure grammar, and transformational grammar, and considers the general mathematical properties of each. Similarly, our performance model should be thought of as an example which illustrates what can be done using a certain kind of process; in other words, we are more interested in the general nature of the method used than in the particular setting, or the particular conversations that the model generates. There are good reasons for this; bearing in mind the immense variety of human conversation, it is clear that similarities in performance can only be of a general and abstract kind. Chomsky showed that it was possible to combine a highly abstract theory with fully explicit examples of its operation in particular cases, and we have tried to emulate him in this respect. So our model is an explicit performance model illustrating a general method of producing conversation.

The best available medium for a performance model of language is the computer program, for three reasons. First, a program must necessarily be formally precise if it is to work at all; second, a computer works out the consequences of a complex model far more quickly and accurately than its programmer; and third, since computers are especially designed to store and process symbolic data-structures, they

provide us with an excellent metaphor and thus a source of new ideas. The program on its own, however, is not an adequate account of the theory it embodies, since it is extremely difficult to divine the overall organisation of a prggram from the raw code. So it is necessary to supplement the program by a full commentary in ordinary English, the program being a sort of guarantee that the ideas really work. It is also necessary to say which parts of the program are intended to model general principles, and which parts correspond only to the particular setting chosen by the programmer. In other words, a program will embody several levels of abstraction, of which only the higher levels are theoretically significant, and the programmer must distinguish clearly between these levels in order to draw attention to what really matters.

Evaluation of Theories

At the end of section 1.1 we gave a list of some phenomena that a theory of conversation ought to account for; for example, it was suggested that a conversational system would have to know that questions were meant to be answered, would have to be able to conduct nested conversations, would have to be able to answer an unexpected question and then return to the right place in its previous train of thought, and so on. Although the list is incomplete, it indicates the kind of criteria by which the program ought to be judged,

since it says what problems it is addressed to. The utterances of the program happen to be grammatical and meaningful, but the processes which ensure this are tricks based on specific properties of the sentences involved, and they could not be generalised. The processes which control the overall conversation, however, could be generalised; indeed, in an earlier version of the program, similar methods were used in another domain in order to produce completely different utterances.

To evaluate the program, then, it is necessary to consider some general properties that a conversational system needs to have, like those mentioned in 1.1, and to see whether the methods embodied in the program could be used in order to give the system the required properties. In his evaluation of various types of grammar, Chomsky rejects those grammars which either cannot describe the language or describe it clumsily; similarly, it can be shown that the methods embodied in a given performance model solve some problems adequately but not others. Our model is analogous to a finite state grammar since there are some respects in which it is clearly inadequate. For example, it cannot give a satisfactory solution to the problem of how people manage to paraphrase a question if it is not understood: the reason is essentially that it has no representation of the effect that an utterance is meant to have on the hearer, and thus no way of modifying its behaviour if the utterance fails to

have the right effect. However, the model does give satisfactory solutions to some of the other problems: for example, it gives a clean and general account of how it is possible to conduct a conversation which is irrelevant to one's previous train of thought, and then to return to one's thoughts at the right place.

The other important criterion, according to Chomsky, is that of learnability; having suggested a set of techniques for conducting conversations, we have to ask whether these techniques could reasonably be learned. The model fares rather badly on this criterion: it cannot learn, and it could not readily be adapted so that it could learn. This issue, along with the other issues of evaluation touched on above, will be discussed more fully in a later section.

To summarise: we want to show in detail how typical human conversations are possible, and we have pursued this aim by writing a formal performance model in the form of a computer program. This model illustrates a particular method of generating conversation, and it is possible to describe the conversation in terms of the structures and processes involved in the production of each part of it. The method we have used gives a satisfactory account of some features of human conversation, but there are some features that it could only explain clumsily, if at all. It could not be readily used in a system which learned to

conduct conversations. All the same, this is not an area in which we can expect to produce an adequate formal theory in one go. We must first get a clearer idea of the properties of different kinds of system, just as Chomsky examines the mathematical properties of different kinds of grammar. Until the pure theory of symbol-manipulating systems is better developed, we cannot expect to get adequate theories in the related areas of Psychology.

1.3 The Setting

The model was designed in accordance with three central features:

(a) The conversation should be conducted by two "robots" (i.e. two programs) and should not involve a human operator.

(b) The robots should inhabit a simple, well-defined universe.

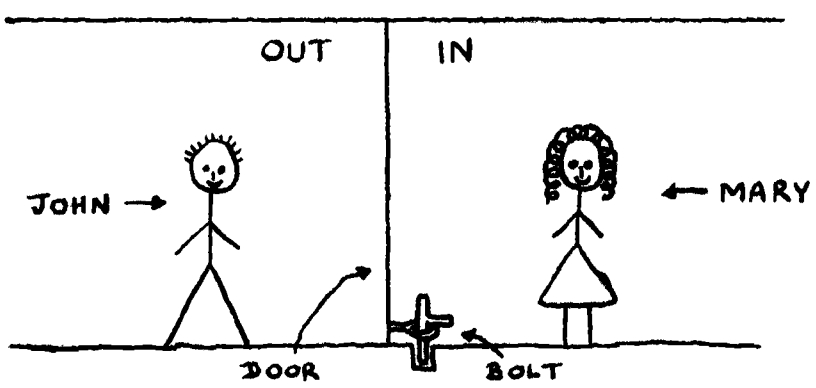
(c) The robots should co-operate in order to achieve a practical goal, the conversation being directed towards this end.

These features will now be explained in more detail.

The decision to have two robots, rather than a robot and an operator, was prompted by the observation that it is usually the human operator that injects purpose into man-machine dialogues. For example, in Winograd's system (which

is reviewed in section 1.5) the program is essentially a slave which responds passively to the questions and commands of the operator; it cannot use language to achieve goals of its own, and if left to its own devices it has nothing to say. By using two robots, and excluding human intervention, we have ensured that the conversation is controlled entirely by the robots, and hence that the method by which it is controlled is fully specified. The robots are called John and Mary, and they are identical in almost all respects, being derived from a general robot program which is compiled twice with different variable names. It is possible to introduce minor differences between John and Mary by initialising their global variables differently; to take a trivial case, each robot has a variable to hold its own name, this variable being set to "John" in one case and "Mary" in the other.

The need to use a simple, well-defined universe in this kind of model is generally recognised. The setting we have chosen is especially simple: it consists of just four objects, each of which can be in one of two positions. The objects are JOHN, MARY, a DOOR, and a BOLT. The two robots can be either IN or OUT; the door can be OPEN or SHUT; and the bolt can be UP or DOWN. The situation might be pictured thus:



There are three actions which a robot can carry out in an attempt to alter the state of the world: the robot can MOVE, or PUSH the door, or SLIDE the bolt. These actions should be thought of as movements which may or may not achieve any result. If an action does achieve something, it moves an object from one of its possible positions to the other. Thus if John employs the action MOVE, and it works, then if he was IN he becomes OUT, and if he was OUT he becomes IN. Similarly, PUSH can make an OPEN door SHUT or a SHUT door OPEN, and SLIDE can make the bolt UP if it was DOWN and DOWN if it was UP. There is no action called PULL to reverse PUSH; if a robot has managed to open the door with PUSH, then a second application of PUSH will close it again.

Whether or not an action succeeds in changing the position of the relevant object depends on the position of one of the other objects, according to the following rules:

(1) If you move when the door is open, you change position.

(2) If you push the door when the bolt is up, the door changes position.

(3) If you slide the bolt when you are in, the bolt changes position.

Thus if you try to move when the door is shut, you remain where you were; and the door only moves when the bolt is up; and you can only move the bolt if you are in. If John (who is out) tries to slide the bolt, one might imagine him carrying out the appropriate movements, but not getting anywhere since the bolt is inside. We are using expressions like "John moves" and "John slides the bolt" in an unusual sense, but there should be no misunderstanding provided that we stick to the bare rules of the universe as defined above, and think of an action as an attempt to induce a change, an attempt which may or may not succeed.

It is assumed that a robot can see all of the objects and carry out any of the actions, unless it is given a special defect - e.g. an inability to see the bolt or to push the door. A robot's ability to see never depends on the state of the world: if John is out and the door is shut, he can still see the bolt provided that he has not been given a special inability to see bolts. So if he cannot see the bolt when he is out, he will not be able to see it when he is in either. Similarly, his ability to push the door depends in no way upon the positions of the door, bolt, or

himself, though the result of his pushing the door obviously depends very much on these positions - in fact entirely on them.

The universe inhabited by John and Mary is thus very simple; it consists of only four objects, three possible actions, and three laws of nature which define the consequences of the actions. The features which appear artificial, such as making all actions equivalent to their opposites, are put in to keep the numbers of actions and laws as low as possible. Instead of enter/exit, push/pull, raise/lower, we use move, push and slide; and we formulate the laws of nature in terms of objects changing position, not in terms of the particular positions they move to. Without this economy, the law that "pushing the door changes its position provided that the bolt is up" would end up as two laws, one for opening the door and one for shutting it. In some universes one would need these extra laws; for example, one could plausibly set up the universe so that a shut door could only be opened if the bolt was up, but an open door could be shut regardless of the position of the bolt. Such an arrangement would certainly be more realistic, but we felt it was more important to keep the model as simple as possible.

Thus our program models a situation in which two robots, John and Mary, occupy a simple world containing two other objects, a door and a bolt; and the robots are the only

active beings - the only beings which can alter the state of the world. The third and last feature of the model is that the robots use language as part of their co-operation to achieve an agreed practical goal. For example, they might agree to try to get the door open, or to get John in, or to get the bolt down. To achieve their goal they will have to agree on a plan, a plan which may require actions from each of them. If, for instance, the goal is to get John in, and the current situation is JOHN OUT, MARY IN, DOOR SHUT, and BOLT DOWN, then Mary will have to slide the bolt up, someone will have to push the door open, and finally John will have to move in. This is an example of a goal which cannot be achieved without co-operation. It is also possible to devise situations where one of the robots could achieve the goal alone, but asks for help because it cannot think of a suitable plan. One can also force a robot to ask for help by limiting what it can see or do; if, for instance, a robot's goal is to open a door that it cannot see, help will clearly be required, if only in the form of answers to questions.

The choice of a co-operative situation was motivated by the observation that language is, on the whole, used for co-operative purposes. A competitive situation, such as a game, does not give rise to conversation so naturally, though the players may co-operate in a discussion of the significance of the game once it is over. The decision to

tie the conversation to the attainment of a practical goal was taken in the belief that this was the easiest kind of conversation to model. When two people exchange news and views, or try to get to know one another, the ensuing conversation is psychologically highly complicated, and there is no straightforward criterion for the relevance of a remark. The existence of a practical goal gives a clear criterion for relevance, and enables us to forget inter-personal subtleties and to concentrate on the psychologically easier processes involved in getting a job done.

We end this section with a summary of the robot's world, in case the reader needs to refer back to it.

(1) The world contains four objects, JOHN, MARY, DOOR, BOLT.

(2) There are three kinds of object: John and Mary are ROBOTS, the door is a DOOR, and the bolt is a BOLT.

(3) A robot must be IN or OUT, a door OPEN or SHUT, and a bolt UP or DOWN. It is possible to have both robots in one place; for instance, they can both be IN.

(4) A robot can try three actions: MOVE, PUSH and SLIDE.

(5) The effects of these actions are as follows:

MOVE changes the robot's position provided that the door is open.

PUSH changes the door's position provided that the

bolt is up.

SLIDE changes the bolt's position provided that the robot is in.

(6) The robots may be limited in what they can SEE and DO. If a robot can see an object, he can always see it. Note that if a robot can move, this only means he can alter his own position; thus John cannot move Mary, or vice-versa.

1.4 Examples of Output

The behaviour of the system will be illustrated by two annotated examples of its output. The first example is a conversation between two robots; and the second is a conversation between a robot and an operator, which enables us to demonstrate how the system responds to unusual utterances.

Example 1

This is a dialogue between the two robots, John and Mary. Each robot is a section of program, and there is a "chairman" function which runs the conversation. The chairman first calls the section of program representing John; John then thinks for a while and eventually returns control to the chairman, usually because he has just made an utterance, and has to give Mary a chance to reply. Then the chairman calls Mary, and it is Mary's turn to think until she wants to say something, whereupon John is

called again, and so on. To set a conversation in motion the operator has to call the chairman function; then the chairman does the rest, returning control only when neither robot has anything more to say.

Before calling the chairman, the operator has to define the starting position by initialising certain key global variables. The items of information that he feeds in are the following:

(a) The positions of the objects.

(b) The objects that can be seen by each robot.

(c) The actions that can be carried out by each robot.

(d) The beliefs of each robot about the consequences of actions.

(e) The names of the robots.

(f) The goals of the robots.

It is here that the differences between John and Mary are introduced - differences both in physical position and state of mind.

In this example, the preliminary situation was defined as follows:

(a) Positions of objects: JOHN OUT, MARY IN, BOLT UP, DOOR SHUT.

(b) John can see all four objects. Mary is blind and cannot see any of them.

(c) John can move, and can slide the bolt, but cannot

push the door. Mary can perform all three actions.

(d) John believes:

1) If you move, nothing happens.

2) If you push the door, it changes position.

3) If you slide the bolt, nothing happens.

Mary believes:

1) If you move when the door is open, you change position.

2) If you push the door, it changes position.

3) If you slide the bolt, nothing happens.

(e) John knows his name is "John" and that his partner is "Mary". Mary vice-versa.

(f) John's goal is to get himself in. Mary has no goal.

To simplify the example, the bolt (and hence the action of sliding it) has been kept out of the dialogue, so that for all intents and purposes the universe could contain only three objects and two actions. We now give a copy of the dialogue printed out by the program from this starting position, with annotations (flanked by stars) at appropriate points.

* Initial state of world is [JOHN OUT MARY IN BOLT UP
* DOOR SHUT].

1 JOHN: MARY.

2 MARY: YES.

3 JOHN: I WANT TO SUGGEST A GOAL.

4 MARY: GO AHEAD.

5 JOHN: WILL YOU HELP ME GET IN.

6 MARY: BY ALL MEANS.

* The robots have now agreed to co-operate to achieve
* John's goal. Mary has learned John's goal, and
* John has learned that she is willing to help. If
* Mary had been given a goal different from John's,
* she would have refused to help him.

7 JOHN: SHALL WE MAKE A PLAN.

8 MARY: JOHN.

9 JOHN: YES.

10 MARY: MAY I ASK YOU SOMETHING.

11 JOHN: GO AHEAD.

12 MARY: ARE YOU IN.

13 JOHN: NO.

* Mary interrupted because she did not know whether the
* goal was already achieved (she is blind, remember);
* had she been cleverer, she would have inferred John's
* position from 5, Before selecting a plan to achieve
* a goal, a robot always checks that the goal is not
* yet attained and that a plan is thus needed. The

* interruption at 8 causes both robots to forget 7:
* John was suggesting a conversation at 7, but Mary
* over-ruled him, and it is she who now starts the
* conversation to choose a plan.

14 MARY: SHALL WE MAKE A PLAN.

15 JOHN: OK.

16 MARY: JOHN.

17 JOHN: YES.

18 MARY: MAY I ASK YOU SOMETHING.

19 JOHN: GO AHEAD.

20 MARY: CAN YOU MOVE.

21 JOHN: YES.

22 MARY: JOHN.

23 JOHN: YES.

24 MARY: MAY I ASK YOU SOMETHING.

25 JOHN: GO AHEAD.

26 MARY: IS THE DOOR OPEN.

27 JOHN: NO.

28 MARY: I SUGGEST THAT WE GET THE DOOR OPEN AND THEN YOU MOVE.

* After John agreed to help choose a plan at 15, Mary
* tried to devise one. To do this she needs to know
* whether the door is open and whether John can move.
* Since she is blind, she has to ask about the door.
* At the point we have reached, Mary has learned the

* following things: that John is out and the door is
* shut; that John's goal is to get in; that John can
* see the door and himself; and that John can move.
* John has learned that Mary cannot see him or the
* floor, and that she is willing to help him. Note
* how Mary's questions are nested in a conversation
* to make a plan.

29 JOHN: MARY.

30 MARY: YES.

31 JOHN: I WANT TO EXPLAIN SOMETHING.

32 MARY: GO AHEAD.

33 JOHN: IF YOU MOVE, NOTHING HAPPENS.

34 MARY: I DISAGREE. IF YOU MOVE WHEN THE DOOR IS OPEN, YOU CHANGE POSITION.

35 JOHN: I SEE.

36 MARY: I SUGGEST THAT WE GET THE DOOR OPEN AND THEN YOU MOVE.

37 JOHN: ALL RIGHT.

* At 29 John has decided that Mary's plan will not
* work, and he interrupts to explain why, giving the
* rule which he used in judging the plan. Mary counters
* by giving the rule on which she had based the plan.
* The robots assume that the more complicated the rule,
* the better, and since Mary's rule contains an extra
* phrase, John accepts it and alters his beliefs.

* Consequently, when she repeats the plan at 36, he
* accepts it.

38 MARY: SHALL WE MAKE A PLAN.

39 JOHN: OK.

40 MARY: I SUGGEST THAT I PUSH THE DOOR.

41 JOHN: ALL RIGHT.

* Mary now pushes the door, and the state of the world
* changes to [JOHN OUT MARY IN DOOR OPEN BOLT UP].
* Since they already agreed as to the consequences of
* pushing doors, there was no discussion of the plan.

42 MARY: I WANT TO TELL YOU SOMETHING.

43 JOHN: GO AHEAD.

44 MARY: I HAVE PUSHED THE DOOR.

45 JOHN: I SEE.

46 MARY: LETS ASSESS THE RESULT OF MY ACTION.

47 JOHN: OK.

48 MARY: NOTHING HAS HAPPENED.

* Since Mary is blind she does not observe a change in
* the position of the door and assumes it is still
* where it was. She is not wise enough to realise
* either that it should have opened, or that she
* should leave the job of assessing her action to
* John, who can see it. In fact the robots were
* deliberately made this way, to provide a circumstance
* in which one robot has to tell the other robot
* something.

49 JOHN: MARY.

50 MARY: YES.

51 JOHN: I WANT TO TELL YOU SOMETHING.

52 MARY: GO AHEAD.

53 JOHN: THE DOOR IS OPEN.

54 MARY: I SEE. THE DOOR HAS CHANGED POSITION.

55 JOHN: YES.

56 MARY: THE DOOR IS NOW OPEN.

57 JOHN: RIGHT.

* Assessing an action consists of two parts: saying
* which object moved, if any, and saying whether the
* action achieved its goal. The first part is done
* in 48 and revised in 54; the second is done in 56.
* The robots do not realise that in this case there
* is no need to say 56 as it follows from 53 and 54.
* After saying RIGHT at 57, John moved, and the
* state of the world has now changed to [JOHN IN MARY
* IN DOOR OPEN BOLT UP].

58 JOHN: I WANT TO TELL YOU SOMETHING.

59 MARY: GO AHEAD.

60 JOHN: I HAVE MOVED.

61 MARY: I SEE.

62 JOHN: LETS ASSESS THE RESULT OF MY ACTION.

63 MARY: OK.

64 JOHN: I HAVE CHANGED POSITION.

65 MARY: JOHN.

66 JOHN: YES.

67 MARY: I WANT TO TELL YOU SOMETHING.

68 JOHN: GO AHEAD.

69 MARY: YOU ARE OUT.

70 JOHN: I DISAGREE. I HAVE CHANGED POSITION.

71 MARY: YES.

72 JOHN: I AM NOW IN.

73 MARY: RIGHT.

* Having decided that the action has achieved its aim,
* the robots now go into a halt state since the main
* goal is achieved and they have no motivation to do
* or say anything further.

Example 2

We now see how the program emerges from a dialogue with a human operator. The dialogue which follows is intended to reveal the defects and virtues of the program in situations which do not usually arise when it is talking to its twin. The operator takes the part of John, and the robot is Mary. The original setting is as follows:

(a) Positions of objects: JOHN OUT, MARY IN, DOOR SHUT, BOLT DOWN.

(b) Mary can see all objects except the door.

(c) Mary can perform all three actions.

(d) Mary believes that whatever you do (move, push, or slide) nothing happens.

(e) She believes she is Mary and her partner is John.

(f) Her goal is to get the door open.

What follows is a copy of an actual dialogue, in which the program printed out the remarks of Mary and I typed in the remarks of John. The annotations are again flanked by stars.

* Initial state of world is [JOHN OUT MARY IN DOOR
* SHUT BOLT DOWN].

1 MARY: JOHN.

2 JOHN: WHAT'S BOTHERING YOU.

* When a robot calls its partner, the reply is ignored,
* so any comment could be put in at 2.

3 MARY: I WANT TO SUGGEST A GOAL.

4 JOHN: GO TO HELL.

* The program analyses English by looking for key words.
* Since "go" appears in expressions like "go ahead" or
* "go on", it is taken as a mark of assent, and Mary
* (at 5) goes ahead with her suggestion.

5 MARY: WILL YOU HELP ME GET THE DOOR OPEN.

6 JOHN: NO.

7 MARY: --

* Having been refused aid, Mary concludes that there
* is no way of achieving her goal, and returns control
* without saying anything.

8 JOHN: MARY.

9 MARY: YES.

10 JOHN: I WANT TO PROPOSE A GOAL.

11 MARY: GO AHEAD.

12 JOHN: WILL YOU HELP ME GET THE DOOR OPEN, EVEN THOUGH I
WAS RUDE.

* The important words here are DOOR OPEN; Mary no more

* notices the apology than she noticed the original
* insult. Nor does she find it strange that John
* has changed his mind; she simply notes that John's
* goal is identical to hers, gives her assent, and
* prepares to co-operate in the attempt to get the
* door open.

13 MARY: BY ALL MEANS.

14 JOHN: THEN LETS MAKE A PLAN.

15 MARY: JOHN.

16 JOHN: WHAT.

17 MARY: MAY I ASK YOU SOMETHING.

18 JOHN: YES OF COURSE.

19 MARY: IS THE DOOR OPEN.

20 JOHN: NO.

21 MARY: SHALL WE MAKE A PLAN.

22 JOHN: YES.

23 MARY: I CANT THINK OF ONE.

24 JOHN: MARY.

25 MARY: YES.

26 JOHN: I OUGHT TO EXPLAIN SOMETHING.

* This interruption (24) would not occur in a conversation
* between robots, but Mary can listen to explanations,
* or answer questions, or be told things, at any time
* during the course of the dialogue. At 23, Mary
* found that according to her theory of how the world
* worked, no action ever caused any change, so she had

* no basis for choosing a plan.

27 MARY: GO AHEAD.

28 JOHN: IF YOU PUSH THE DOOR WHEN THE BOLT IS UP, THE DOOR ALTERS POSITION.

29 MARY: I SEE.

30 JOHN: MARY.

31 MARY: YES.

32 JOHN: LET ME EXPLAIN SOMETHING ELSE.

33 MARY: GO AHEAD.

34 JOHN: IF YOU MOVE, NOTHING HAPPENS.

35 MARY: I ALREADY KNOW THAT.

36 JOHN: MARY.

37 MARY: YES.

38 JOHN: I WANT TO EXPLAIN ONE MORE THING.

39 MARY: GO AHEAD.

40 JOHN: IF YOU SLIDE THE BOLT, IT CHANGES POSITION.

41 MARY: I DISAGREE.

42 JOHN: --

43 MARY: IF YOU SLIDE THE BOLT, NOTHING HAPPENS.

44 JOHN: I DISAGREE.

* The three explanations have different outcomes, In
* the first (28) John's rule is better (i.e. it has a
* proviso which Mary's lacks) and Mary accepts it
* without further ado; in the second (34) the rules
* are identical; and in the third (40) they are different
* but equally good (neither has a proviso), and neither

* party changes its mind. Having accepted John's
* first rule (28), Mary can now formulate a plan.

45 MARY: I SUGGEST THAT WE GET THE BOLT UP AND THEN I PUSH THE DOOR.

46 JOHN: OK.

47 MARY: SHALL WE MAKE A PLAN.

48 JOHN: YES.

49 MARY: I SUGGEST THAT I MOVE.

50 JOHN: TOODLE PIP.

* At 49 Mary has no proper basis for making a plan,
* but since her beliefs tell her that actions sometimes
* get you somewhere she suggests an arbitrary plan.
* John's reply should have been a reaction to the plan,
* and Mary judges it inappropriate. The robots react
* to inappropriate remarks by declaring that a muddle
* has arisen and returning to an earlier stage in
* their thoughts; in this case Mary returns to the
* stage she had reached at 14. Although she goes
* back, the conversation is not repeated because her
* changes in belief are retained.

51 MARY: JOHN.

52 JOHN: YES.

53 MARY: WE HAVE GOT MUDDLED; LETS START AGAIN.

54 JOHN: OK.

55 MARY: SHALL WE MAKE A PLAN.

56 JOHN: YES.

57 MARY: I SUGGEST THAT WE GET THE BOLT UP AND THEN I PUSH THE DOOR.

58 JOHN: ALL RIGHT.

59 MARY: SHALL WE MAKE A PLAN.

60 JOHN: OK.

61 MARY: I SUGGEST THAT I SLIDE THE BOLT.

62 JOHN: GOOD IDEA.

* The plan in 61 was chosen arbitrarily, like the one
* in 49, but in this case she has a lucky hit, and her
* action changes the state of the world to [JOHN OUT
* MARY IN DOOR SHUT BOLT UP].

63 MARY: I WANT TO TELL YOU SOMETHING.

64 JOHN: GO AHEAD.

65 MARY: I HAVE SLID THE BOLT.

66 JOHN: VERY GOOD.

67 MARY: LETS ASSESS THE RESULT OF MY ACTION.

68 JOHN: OK.

69 MARY: THE BOLT HAS CHANGED POSITION.

70 JOHN: RIGHT.

71 MARY: THE BOLT IS NOW UP.

72 JOHN: YES.

73 MARY: I WANT TO EXPLAIN SOMETHING.

74 JOHN: GO AHEAD.

75 MARY: IF YOU SLIDE THE BOLT WHEN YOU ARE IN, THE BOLT

CHANGES POSITION.

76 JOHN: I SEE.

* If an action gives rise to a consequence different
* from that predicted by your rule, you change the
* rule. Mary assumes that some feature of the
* current situation must be the "proviso" which
* enabled her action to work, and has a choice between
* the door being shut and she being in; her choice
* is arbitrary and hence a lucky hit. Having formulated
* this new rule, she checks her model of John's beliefs
* to see whether he believes it; since she has recorded
* that he believes something else, (see remark 40),
* she explains her new rule (75). Now she returns to
* the plan, and pushes the door, which opens; the
* main goal is thus achieved, and we will break off
* the conversation at this point.

1.5 Related Work

1.5.1 Previous conversational programs

In this section, we review five recent conversational systems, those of Bobrow (1964), Weizenbaum (1966), Colby et al (1970), Carbonell (1970), and Winograd (1972). We are interested in two aspects of language use: first, how language is understood, and second, how interactions are controlled. The programs mentioned above were chosen because they all have something to say on one or both of these aspects, and in reviewing them we will ignore any other features that they have. For example, there will be no discussion of how the programs deal with syntax, although Winograd's program, for example, has a great deal to say on this matter.

In recent years, the problem of building semantic descriptions has become the major concern of research in Artificial Intelligence, (see Minsky, 1968, and Minsky and Papert, 1972), and the task of understanding a sentence is thus seen as the task of building a semantic description of it. A semantic description is a representation of the input in a form useful to the system; thus the nature of the description will depend on the system's goals. A useful description of the input needs to have three features:

it must be expressed in a standardised format; it must ignore those parts of the input which are irrelevant; and it must include any relevant information not contained in the input, this information being drawn from the system's previous knowledge. The programs of Bobrow, Carbonell and Winograd all use semantic descriptions, each description being adapted to the purposes of the program in question. The programs of Weizenbaum and Colby, on the other hand, are not concerned with semantic description, but are included because they are highly interactive: indeed, they have both at different times passed the "Turing Test" (that is, fooled human observers into thinking that their output was produced by a human being).

Bobrow's STUDENT

We begin with one of the first systems to use a semantic model, Bobrow's STUDENT, which is described in his thesis "Natural Language Input for a Computer Problem-Solving System" (1964). The heart of Bobrow's system is a set of programs for solving linear equations, and he is thus able to use the equation format used by these programs as his semantic representation. The main job of his program is to translate a paragraph of natural English into a set of equations in the prescribed format. Thus the program only accepts a limited subset of English, drawn from the types of expression an examiner would use in phrasing an algebra problem. Having converted the input

into a set of equations, the system tries to solve them, using any other equations it knows which are relevant (e.g. it might already know such general relationships as Distance = Speed x Time). If it finds thats its equations are insufficient for a solution, it is able to request further knowledge and to use it if it turns out to be relevant. Here is an example of a man-machine interaction, adapted from Bobrow in "Semantic Information Processing", (Minsky, 1968) page 180.

1. Operator: THE PROBLEM TO BE SOLVED IS - THE GROSS WEIGHT OF A SHIP IS 20,000 TONS. IF ITS NET WEIGHT IS 15,000 TONS, WHAT IS THE WEIGHT OF THE SHIP'S CARGO?
2. Student: DO YOU KNOW ANY MORE RELATIONSHIPS AMONG THESE VARIABLES - GROSS WEIGHT OF SHIP; TONS; ITS NET WEIGHT; WEIGHT OF SHIP'S CARGO.
3. Operator: YES.
4. Student: TELL ME.
5. Operator: THE WEIGHT OF A SHIP'S CARGO IS THE DIFFERENCE BETWEEN THE GROSS WEIGHT AND THE NET WEIGHT.
6. Student: THE WEIGHT OF THE SHIP'S CARGO IS 5,000 TONS.

After the first utterance in this dialogue, STUDENT has formed two equations, akin to x = 20,000 and y = 15,000, where x is the gross weight and y the net weight. It gives the cargo weight the letter z, perhaps. Then it

cannot get any further because it lacks the relationship z = x - y. So it lists the variables it wants related, and at utterance 5 receives the equation it needs to complete the solution.

The remarks printed out by STUDENT fall into two categories: those which are merely "thinking aloud", and those which require a reply from the operator. The first kind are only used to show the operator what is going on, and have been omitted from the example above. The genuinely interactive remarks are of four kinds (corresponding to the points in the algorithm where they occur): specifications of what information is needed; requests for information; solutions; and failure reports. The first three are illustrated by utterances 2, 4 and 6 respectively, in the example. A solution or failure report ends the dialogue, but the other two kinds of remark can lead to a variety of conversations depending on the operator's reply. Utterance 2 in the example is a yes-no question, and precedes a 2-way decision node in the algorithm. If the answer is "yes", the system prints "TELL ME" in order to obtain the necessary information; if the answer is no, it prints a failure message. The course of the dialogue after utterance 5 (i.e. the reply to "TELL ME") depends on whether the information did the trick. If it did, the solution is printed; if not, the system loops back to

utterance 2, and respecifies the information it needs. This process continues until a solution is found or the operator says "no" at utterance 2.

The goal of STUDENT is clearly to find the solution of its current algebra problem; this goal is implicit in the program in the sense that it is the only goal STUDENT is able to tackle. Its semantic description of English sentences enshrines exactly the features we mentioned above: the description is oriented to the goal of the system and the particular procedures it uses to achieve the goal. In other words, the input is transformed into a linear equation in a format which the solving procedures can handle. Furthermore, the description ignores much of the information in the surface form, replacing expressions such as "THE WEIGHT OF THE SHIP'S CARGO" by variable names. The system is capable of interacting with the user, but its conversational repertoire is limited to two or three snatches of dialogue, and there are no facilities for embedding one conversation in another or making an interruption. It is the translation of an inter-related piece of discourse into a semantic description which constitutes the chief purpose of the system.

Weizenbaum's ELIZA

As a contrast to Bobrow's program, it is worth mentioning two systems which can take part in a wider range

of conversations, but which lack STUDENT'S well-developed semantic model. The systems represent the two parties in a psychotheraputic encounter: the therapist is ELIZA, created by Weizenbaum (1966), and the patient is the paranoid model of Colby, Weber and Hilf (1971).

ELIZA practises what is known as "Rogerian psychotherapy", in which the aim of the therapist is to reflect back the utterances of the patient rather than to interpret them. The patient then feels understood and accepted, and is encouraged to go further with his mental excavations. Weizenbaum realised that this process of drawing out the patient by reflecting back his remarks could to some extent be carried out by purely syntactic manipulations. ELIZA has two kinds of rules, D-rules for decomposing sentences and R-rules for re-assembling them. On receiving an input sentence, the program searches it for key words; if it finds a key word, it decomposes the sentence according to a D-rule associated with that particular key word. Each D-rule has several R-rules associated with it, and one of these is used to assemble a new sentence, the sentence which becomes the output. For example, the sentence "I know you hate the sight of me" might be analysed by a D-rule into the pattern x + YOU + y + ME where x is "I KNOW" and y is "HATE THE SIGHT OF", then transformed by the R-rule WHAT MAKES YOU SAY I + y + YOU into the output, "What makes

you say I hate the sight of you". If ELIZA cannot find a key word in the input, she falls back on a store of psychotheraputic platitudes such as "Tell me something about your mother", or just "Please go on".

ELIZA is chiefly notable for the clever way in which she disguises her complete lack of comprehension. She can keep a conversation going by rearranging the patient's sentences, or change the topic of conversation by introducing a sentence from her store of platitudes, but her semantic content is nil. In his analysis of Bobrow's STUDENT, Dreyfus (1972) argues that it is highly misleading to claim that STUDENT "understands English", and one can see what he means; all the same, there is a clear distinction between the way English is treated by STUDENT and by ELIZA.

Colby's Artificial Paranoid

The paranoid model of Colby et al (1971) is an attempt to simulate the behaviour of an actual patient in an interview. The model is by no means the puppet of its operator: it has its own motivation and is highly emotional. Its emotional state is represented by three variables, corresponding to fear, anger, and mistrust. It has several delusions concerning the underworld and the Mafia, and is motivated to express these delusions provided that its mistrust of the operator (i.e. the therapist) is not too high. Thus its conversation is a compromise between

expressing its own thoughts and responding to the promptings of the operator. Both input and output sentences have inner representations in terms of their emotional significance, and these representations are paired to give a set of input-output associations. Thus an input containing a reference to a sensitive personal topic (e.g. sex or gambling or family) provokes a defensive reaction; or an angry input provokes a hostile reaction; and so on. Not all the input-output pairings are this emotional, however: a straightforward question (e.g. "What is your name?") will normally elicit an answer; and if the operator touches on a delusion topic, the program will jump at the opportunity to express its delusions, despite an initial reaction of fear.

The model is able to conduct a reasonably convincing and coherent dialogue because its knowledge of the topics likely to arise is stored in terms of suitable replies to the most probable questions. Its understanding is thus not very deep or wide, but if it fails to understand an input it can react by producing a fresh delusion, or even by returning control without printing anything - that is, by keeping quiet. It seems to be able to refer back to earlier remarks in the dialogue in order to interpret pronouns, but in general it is the operator that makes the conversation hang together; like ELIZA, the paranoid

model is primarily concerned with responding to the operator's last remark, and has no higher-level representation of the conversation as a whole.

Carbonell's SCHOLAR

The three programs we have examined so far were written for different reasons: Bobrow was primarily interested in language, Colby in paranoia, and Weizenbaum's ELIZA was a semi-humorous attempt to write a simple program which appeared to understand English, but really cheated. The next project we shall consider is SCHOLAR (Carbonell, 1970) which was written in order to extend the techniques of Computer-Aided Instruction (CAI). Carbonell claims that previous systems in CAI worked by using detailed and specific "frames" of data which the teacher had to feed in before each session. In other words, the program had to be told the specific questions it was going to ask, and their answers; it could not devise its own questions or answer unexpected questions from its pupil. SCHOLAR gets round these limitations by having a semantic model of its subject, together with a facility for conducting "mixed-initiative" dialogues - dialogues in which both parties can ask questions, and the user can interrupt the program. SCHOLAR gives lessons in the geography of Latin America, and the following is a sample dialogue (adapted from Carbonell et al, 1973).

1. Scholar: WAHT IS THE LANGUAGE OF ARGENTINA?
2. Pupil: PROBABLY SPANISH.
3. SŏHolar: VERY GOOD. APPROXIMATELY WHAT IS THE AREA OF ARGENTINA?
4. Pupil: TELL ME SOMETHING ABOUT PERU.
5. Scholar: PERU IS A COUNTRY. IT IS LOCATED IN SOUTH AMERICA. THE CAPITAL OF PERU IS LIMA. NOW ANSWER THE QUESTION YOU DIDN'T ANSWER BEFORE.
6. Pupil: THE AREA OF ARGENTINA IS 1,100,000 SQUARE MILES.
7. Scholar: APPROXIMATELY, BUT A MORE EXACT VALUE IS 1,200,000. THE EASTERN BORDERING COUNTRY OF PERU IS ...
8. Pupil: CATHOLICISM.
9. Scholar: WRONG. CATHOLICISM IS NOT A COUNTRY BUT A RELIGION. THE CORRECT ANSWER IS: BRAZIL. WHERE IS PERU?
10. Pupil: IS ASUNCION HOT?
11. Scholar: SINCE ASUNCTION IS PART OF PARAGUAY, YES.

This example illustrates a number of features, but we will follow up only three of them here. First, it is evident that the system does know something about the meaning of the concepts it is using (note its reply to utterance 8) and it is worth looking briefly at how this semantic information is stored. Second, it is clearly not picking out questions at random: it begins with the

topic of Argentina (1,3); then it switches to Peru (7,9) when the pupil expresses ignorance on the subject (4). So we want to know how it arganises its questions in this systemmatic way. Thirdly, we note that it is possible for both parties to ask questions; at utterance 4, for example, the pupil interrupts and asks about Peru, and the program first responds to his request, then tells him to answer question 3, and is able to interpret the answer in spite of the intervening dialogue. How is this ability to handle interruptions and nested conversations achieved?

The semantic model of SCHOLAR is based on the work of Quillian (1967). Quillian uses what he calls a "semantic network" to store both factual and conceptual information. The network takes the form of nodes connected by links, the nodes representing concepts, such as "latitude", "capital", or "South America", and the links representing relationships between them. In fact it is more complicated than this, since some concepts can be both nodes and links. Some links are especially important in the SCHOLAR system, and we shall mention four: SUPER-CONCEPT, SUPER-PART, EXAMPLES and APPLIED-TO.

The most familiar of these links is EXAMPLES, which would link a node such as COUNTRY with nodes like ARGENTINA, PERU, CHILE, and so on. SUPER-CONCEPT is its inverse:

it would link SANTIAGO with CAPITAL and CITY and PLACE. The link SUPER-PART only really applies to places: it links an area with the larger area which encloses it (e.g. SANTIAGO with CHILE and SOUTH AMERICA). Finally, APPLIED-TO tells you what a given concept can be applied to, and thus links (say) CAPITAL with STATE, PROVINCE and COUNTRY.

To illustrate the structure of the SCHOLAR data base, we give a few entries from it, greatly over-simplified and adapted from Collins, Carbonell and Warnock (1972).

CAPITAL

SUPER-CONCEPT: CITY

APPLIED-TO: COUNTRY, STATE, PROVINCE

EXAMPLES: BUENOS-AIRES, SANTIAGO, ASUNCION

ARGENTINA

SUPER-CONCEPT: COUNTRY

SUPER-PART: SOUTH AMERICA

CAPITAL: BUENOS-AIRES

BUENOS-AIRES

SUPER-CONCEPT: CITY, CAPITAL

SUPER-PART: ARGENTINA, PAMPAS, SOUTH AMERICA

It is a feature of this kind of network that it contains a mixture of conceptual and factual information; for example, "a capital is a city" is conceptual, while "Buenos-Aires is the capital of Argentina" is factual. So

SCHOLAR uses the network both to answer questions (e.g. utterance 11) and to detect meaningless remarks (e.g. 9).

Our second question concerned the way in which SCHOLAR generates related questions. Generating a question involves three major decisions: (a) the choice of context; (b) the choice of question content within that context; (c) the choice of the mode of presentation of the question. Possible contexts are ARGENTINA, PERU, CHILE, etc - we will suppose the system chooses ARGENTINA. Next it has to decide the question content, and it represents this by strings such as BUENOS-AIRES CAPITAL ARGENTINA or COUNTY SUPER-CONCEPT ARGENTINA. Finally it decides on the question format, e.g. "What is the capital of Argentina?" or "The capital of Argentina is ..." or "The capital of Argentina is Santiago - right or wrong?" or "Is it correct to say that Argentina is not a country?".

The choices of the exact question content and the presentation format are made probabilistically; that is they are partly arbitrary, but varied to ensure that a question is not repeated and that a particular format is not overworked. It is the first choice, that of context, which reflects the overall strategy of the system and its model of the pupil. Before a teaching session, SCHOLAR can be given a context in which it is meant to operate (SOUTH AMERICA, say). It then selects a sub-context

(ARGENTINA) and asks one or two questions within it in order to test its pupil's knowledge. If the pupil answers correctly, SCHOLAR will pass more quickly on to a fresh context (CHILE, say). It uses any clues picked up in the dialogue to discover areas where the pupil is ignorant, and thus switches to PERU (in the sample dialogue) after getting a question on that topic. SCHOLAR keeps track of the time left in the lesson, and the breadth of the topic it is meant to teach, and plans its context changes accordingly.

Finally, we consider the way in which SCHOLAR interacts with the user. It has two primary modes of interaction, the TEST mode and the Q/A mode. In the TEST mode, the program asks questions to the pupil and comments on his answers, ignoring interruptions and questions. The Q/A mode is the other way round: the pupil asks questions and SCHOLAR answers them. The mixed-iniative mode is a combination of these simpler modes. If SCHOLAR is started in the mixed-iniative mode, as it normally is, it begins by calling the TEST block of program and asking questions. But it also tests each input from the pupil to see whether it is a question or not; and if the pupil does interrupt with a question, the question is passed to the Q/A block together with a message indicating that it should answer just this one question and then return control. When control returns, a message is typed out telling the pupil to answer the previous question, and the TEST block continues

from where it left off.

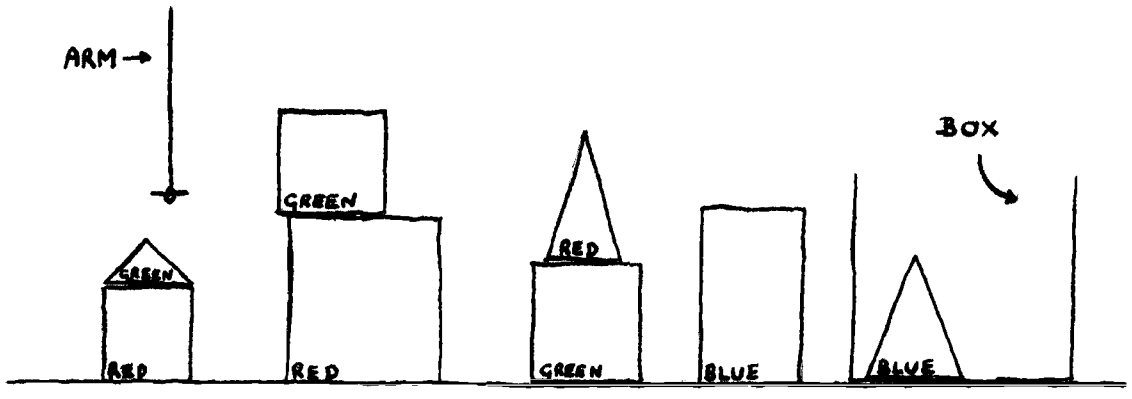
It is possible for the user to decide which mode he wants to use: he can choose between Q/A and mixed-initiative. If, for example, the mode is currently mixed-initiative, and he wants to ask a series of questions, the user types Q/A, and SCHOLAR goes over to Q/A mode and stops asking test questions. Thus the general form of interaction can be decided by agreement, as it were.

To sum up: SCHOLAR is an attempt to write a CAI system using a semantic model based on the network method of Quillian. It can carry out two kinds of conversational process in a subset of English: asking questions, and answering them. There are two blocks of program corresponding to these two processes, and the system has a facility which enables it to jump from one block to the other and back again when it is in its mixed-initiative mode, thus achieving a nested section of conversation. When it is asking test questions, the questions are loosely related in that they share a common context, and this introduces an element of coherence into the dialogue. The goal of SCHOLAR is to explore the topic it is asked to teach, with particular emphasis on those areas in which the pupil performs least well.

Winograd's "Understanding Natural Language"

The last program we consider is perhaps the most impressive. It was written by Winograd (1972) and is

notable for the way in which it carries out syntax, semantics and inference in principled ways, and relates the three processes. Thus it is able to use semantic considerations to guide the parsing, and to use inference in order to determine meaning. The setting is a toy world containing blocks and pyramids of various colours and sizes, a box, and an arm which can pick up the blocks and pyramids and move them to new positions. Here is a short extract from the system's output, illustrating its ability to understand questions and commands. The original setting is this:



1. Operator: PICK UP A BIG RED BLOCK

(The program picks up the green block, puts it down in an empty space, then picks up the red block which it can now get at.)

2. Program: OK.

3. Operator: GRASP THE PYRAMID.

4. Program: I DON'T UNDERSTAND WHICH PYRAMID YOU MEAN.

5. Operator: FIND A BLOCK WHICH IS TALLER THAN THE ONE YOU ARE HOLDING AND PUT IT IN THE BOX.

(The program uses inference to find out which block it is holding, and works out that "it" refers to a taller block, namely the blue block. It then releases the red block and puts the blue block in the box.)

6. Program: OK.

7. Operator: WHAT DOES THE BOX CONTAIN?

8. Program: THE BLUE PYRAMID AND THE BLUE BLOCK.

9. Operator: HOW MANY BLOCKS ARE NOT IN THE BOX?

10. Program: FOUR OF THEM.

11. Operator: IS AT LEAST ONE OF THEM NARROWER THAN THE ONE WHICH I TOLD YOU TO PICK UP?

12. Program: YES, THE RED CUBE.

This example demonstrates that the program is able to carry out two high-level operations: inferring, and planning. It uses inference in order to understand questions (5,11), and also to answer them (9); and it uses planning to carry out commands (1). These operations are conducted in a language called PLANNER, which was designed by Hewitt (1971) with this kind of use in mind. Winograd uses PLANNER to represent the meanings of the input sentences, and also as a medium for all sorts of general purpose information, some conceptual and some factual, which the system contains

at the onset. To understand the meaning of an input sentence, then, the program converts it into the appropriate PLANNER expression. The way in which this is done is an important aspect of Winograd's system, involving a full parsing of the input in terms of a systemic grammar; but since we are not concerned here with syntax, we will forget about how the PLANNER expressions are arrived at, and concentrate instead on what they look like and how they are used.

The PLANNER data base can store information in two forms: as ASSERTIONS, or as THEOREMS. Roughly speaking, assertions are used to express simple propositions such as "The block is green", and theorems are used to express more complex propositions such as "All blocks are objects". An assertion is a list of symbols in some standard order, usually beginning with a predicate. This list is often called a PATTERN, the implication being that it is an arrangement of several constituents in which the order is important. Here are some English sentences translated into possible PLANNER assertions. In illustrating PLANNER expressions we will not use the notation of Winograd or Hewitt, but will rely on a simpler, adapted form. The symbols "BLOCK2", "PYRAMID3", etc are used as proper names to refer to objects.

"The small block is green" [COLOUR BLOCK3 GREEN]

"Green is a cplour" [IS GREEN COLOUR]

"The small block is on the big one" [ON BLOCK3 BLOCK4]

"The big block is under the small one" [ON BLOCK3 BLOCK4]

There are two important points to note here. First, the assertions represent the meanings of their English equivalents in a concise and standardised format. Referring expressions are replaced by their referents, and paraphrases are reduced to an exactly similar form. There is thus a distinct advantage in working with the PLANNER expressions rather than the English ones. Second, it is possible for PLANNER assertions to represent several different types of information. The most natural grouping is into three types of assertion, each of which is represented above: [IS GREEN COLOUR] is a conceptual assertion, which would be true in all worlds; [COLOUR BLOCK3 GREEN] is a factual assertion which is always true in this world; and [ON BLOCK3 BLOCK4] is a factual assertion which may be true at some times and false at others. The system's model of the current state of the world is mainly composed of the set of assertions of this last kind.

The other expression used in PLANNER is, as we said above, the THEOREM. Theorems can represent various kinds of more complicated knowledge, such as the following:

(a) All blocks are objects.

(b) A wompom is a red block in the box.

(c) If you want to pick up a block, clear its top and then grasp it.

(d) A block cannot be both on top of and underneath another block.

The basic philosophy of PLANNER is that in order to decide how to represent these propositions, you begin by asking how they are going to be used. How, for example, does one use a proposition of the form "All blocks are objects"? The answer is, of course, that one uses it as the first premise of a syllogistic inference: e.g. "All blocks are objects, X is a block, therefore X is an object". Thus one uses it to derive the proposition "X is an object", and completes the derivation by finding another premise "X is a block". In PLANNER, therefore, the sentence "All blocks are objects" is given the following interpretation: "If you want to prove something is an object, prove it is a block and you're through." Similarly, the other propositions (b), (c) and (d) would be transcribed as follows:

(b) If you want to prove something is a wompom, first prove it's red, then prove it's a block, then prove it's in the box.

(c) To achieve the goal of picking up a block, first achieve the goal of clearing its top, then grasp it.

(d) If you want to assert that block X is on top of block Y, erase from the data base any assertion which says Y is on X.

PLANNER has three kinds of theorem with which to represent these propositions, and we will call them INFER-THEOREMS, ACHIEVE-THEOREMS, and ASSERT-THEOREMS. This classification reflects the purposes of the theorems; their manner of operation is basically similar. To show how theorems work, and what they look like, we will consider a theorem for "All blocks are objects":

```
INFER-THEOREM T1 [IS ?X OBJECT]
PROVE [IS ?X BLOCK]
END
```

This theorem has been given the name T1, and it is a procedure. But unlike most procedures, it is not usually invoked by calling its name. In PLANNER, a procedure call is not of the form "Call procedure X", but rather, "Call some procedure to achieve purpose X", and the purpose of a procedure is indicated by its opening pattern. In the case of T1, this pattern is [IS ?X OBJECT], meaning that the purpose of T1 is to infer that something is an object. The symbol "?X" is a variable, this being marked by the use of the prefix "?". The PLANNER data base will contain a number of theorems, each with a different name, but the patterns which introduce different theorems can be the same; for instance, the theorem which means "All pyramids are objects" will also begin with the pattern

[IS ?X OBJECT]. Thus the PLANNER system invokes a procedure by consulting its purpose, and can have several different procedures to achieve the same purpose.

Suppose that the data base contains only T1 together with the assertions [IS BLOCK3 BLOCK] and [IS BLOCK4 BLOCK], and that we want to prove the assertion [IS BLOCK4 OBJECT]. To do this, we use the command PROVE[IS BLOCK4 OBJECT], and the system begins by taking this pattern and MATCHING it with every assertion in the data base. Matching is the process of comparing two patterns symbol by symbol, and a match only succeeds if every symbol in pattern A is identical to the corresponding symbol in B, variables being allowed to adopt any chosen value. In this case, the "target" pattern [IS BLOCK4 OBJECT] fails to match either assertion in the data base, since OBJECT is not identical to BLOCK, and the system then looks at the INFER-THEOREMS and tries to find one whose opening pattern matches the target. The opening pattern of T1 is [IS ?X OBJECT], and this does in fact match [IS BLOCK4 OBJECT], ?X picking up the value "BLOCK4". The success of this match means that T1 will now be run, and that ?X will keep the value "BLOCK4" throughout its execution.

T1 contains only one command in its body, the

instruction PROVE [IS ?X BLOCK]. Since ?X now has a value, this is read as PROVE [IS BLOCK4 BLOCK], and the system carries on as before, trying to find an assertion to match this new target. Since the assertion [IS BLOCK4 BLOCK] is one of the ones in the data base, it succeeds, and since there are no more instructions in T1, T1 succeeds as well, and the inference is made.

Let us examine another case. Suppose that instead of beginning with the command PROVE [IS BLOCK4 OBJECT], we had said PROVE [IS PYRAMID2 OBJECT]. The computation would have been similar until the match with the pattern at the head of T1; this match would succeed again, but this time ?X would get the value "PYRAMID2", and the new target would be [IS PYRAMID2 BLOCK]. There is no assertion matching this in the data base, nor any theorem matching it, so it would fail, and so would T1. However, the failure of T1 is not the end of the process: the system might find another theorem matching [IS PYRAMID2 OBJECT], and this theorem might succeed. It is only when it finds that no other such theorem exists that the system gives up and fails to make the inference. To succeed, it would have needed an extra theorem meaning "All pyramids are objects", and the extra assertion [IS PYRAMID2 PYRAMID].

Another example of an infer-theorem is that for proposition (b), "A wompom is a red block in the box". This

might be represented by the following procedure:

```
INFER-THEOREM T2 [IS ?X WOMPOM]
PROVE [IS ?X BLOCK]
PROVE [COLOUR ?X RED]
PROVE [IN ?X BOX]
END
```

This theorem could be used to infer [IS BLOCK6 WOMPOM], for instance, and for this inference to succeed, all three sub-inferences inside T2 would also have to work out.

Proposition (c), "If you want to pick something up, clear its top and grasp it", would be represented by an ACHIEVE-THEOREM. This is slightly different from an infer-theorem, the point being to bring about the target rather than to infer that it has already been brought about. To use the PLANNER system to achieve things, you specify a number of basic actions, and then write ACHIEVE-THEOREMS which organise appropriate sequences of these actions. Winograd uses just three basic actions, GRASP, MOVETO and UNGRASP. The PLANNER theorem for (c) might look like this:

```
ACHIEVE-THEOREM T3 [PICKUP ?X]
GOAL [CLEARTOP ?X]
PROVE [AT ?X ?Y]
GOAL [MOVETO Y?]
```

```
GOAL [GRASP]
END
```

This procedure includes a sub-goal, an inference to pick up information, and two basic actions. Suppose we begin with the command GOAL [PICKUP BLOCK2]. The system matches this with [PICKUP ?X] and ?X gets the value "BLOCK2". Then the goal [CLEARTOP BLOCK2] is tackled, and the system looks for another achieve-theorem matching this new target. Suppose it succeeds (if it fails, T3 fails and a new theorem is sought); the next instruction then reads PROVE [AT BLOCK2 ?Y] and is equivalent to the question "Where is BLOCK2?". In inferring this target, the system picks a value for ?Y which is the position of BLOCK2; it then uses this value as argument for the basic action MOVETO, and thus gets into position to grasp BLOCK2. (It must be stressed that this account is highly oversimplified, and that we are making no effort to follow the details of Winograd's implementation. For an authentic account of the above theorems the reader is referred to Winograd (1972), sections 6 and 7).

The last proposition mentioned above was (d): "If Block X is on Block Y, Y cannot be on X." This is represented by the other kind of theorem, an ASSERT-THEOREM, as follows:

```
ASSERT-THEOREM T4 [ON ?X ?Y]
ERASE [ON ?Y ?X]
END
```

T4 is invoked whenever an assertion such as [ON BLOCK2 BLOCK3] is added to the data base. To add an assertion, one says ASSERT [ON BLOCK2 BLOCK3], and this has two effects: first, the pattern is put in the data base; second, all ASSERT-THEOREMS matching the pattern are run. When T4 is run, [ON BLOCK3 BLOCK2] (if it exists) is erased by the body of T4, and the data base is thus cleansed of a possible logical contradiction.

It should now be evident how the PLANNER system represents statements, questions, and commands. Statements such as "The green pyramid is on the red block" are converted to ASSERTIONS, like [ON PYRAMID2 BLOCK2], the assertion is put into the data base, and all matching ASSERT-THEOREMS are run in order to clean up any logical clashes with other assertions. More complex statements such as "All blocks are nice" or "A steeple is a green pyramid" are converted into INFER-THEOREMS which are put in readiness in the data base, but not yet run. Questions, such as "Is the block green", or "Which object is in the box" are converted into patterns, and the answer is sought by inference, following an instruction of the form PROVE [COLOUR BLOCK4 GREEN] or

PROVE [IN ?X BOX]. Finally, commands such as "Put the blue pyramid in the box" are carried out by means of ACHIEVE-THEOREMS invoked by calls of the form GOAL [IN PYRAMID2 BOX].

We have given a quick, schematic sketch of PLANNER, with the aim of conveying an intuitive idea of how it works; we now conclude our examination of Winograd's program by looking briefly at the way in which it interacts with the user.

The dialogue between Winograd's program and its operator consists essentially of a series of pairs: the operator says something, and then the program makes an appropriate response. The program almost never takes the initiative and the overall structure of the conversation is controlled by the operator. There are three inputs that the operator can make: statement, question, command; and the range of possible responses is as follows:

(a) If asked a question, the program normally answers it, making intelligent use of its knowledge of the world and the previous dialogue in order to choose suitable referring expressions. If it cannot give a straight answer, the program will respond with "I DON'T KNOW" or "I DON'T UNDERSTAND". or "I'M NOT SURE WHAT YOU MEAN BY ...DO YOU MEAN 1)... OR 2)...?" This last response requires the operator to type a 1 or 2 to

disambiguate a phrase, and his answer can only be interpreted in this form.

(b) To a statement, the program can reply "I UNDERSTAND" or "I DON'T UNDERSTAND ...", or (presumably) by "I'M NOT SURE WHAT YOU MEAN BY ..."

(c) To a command, it can reply "OK" if it carries out the command, or "I CAN'T", or "I DON'T UNDERSTAND...", or "I'M NOT SURE WHAT YOU MEAN BY ..."

The program cannot conduct a conversation more complicated than one of these pairs; for instance, it cannot ask proper questions or conduct nested conversations. The "I'M NOT SURE WHAT YOU MEAN BY ..." expression is a special case; its reply is not parsed and not converted into a PLANNER expression. The program keeps a record of previous utterances and events in order to interpret pronouns and referring phrases, but it does not organise its record of the dialogue according to any higher-level interpretation, nor does it have any expectancies about the operator's forthcoming remarks.

The implicit goal of the system is to respond to the promptings of the operator; it has no explicit overall goal (i.e. no goal that could be altered). It does however have temporary goals supplied by the commands of the operator, and it achieves them in a principled way by building the sub-goal tree which arises from the operation

of PLANNER procedures. But it cannot use language as an aid to achieving its goals. There is no facility, for example, enabling it to interrupt the planning process, ask a question (about whether a certain subgoal is achieved, perhaps) and then return to its plans and make use of the answer.

To sum up: Winograd's program is essentially a language understanding system, rather than a language producing system; consequently, its output is geared to demonstrating that it understood the input, and is not related to any other goals. Winograd's main purpose was to show how it is possible to make use of pre-existing knowledge in order to interpret fresh utterances, and he represents meaning and knowledge in PLANNER expressions. The PLANNER data base is a loosely organised collection of assertions and procedures, the procedures being invoked by pattern-matching and not by a direct call of the procedure's name; consequently, it is easy to add new knowledge; new assertions or theorems just have to be put somewhere in the data base. Using PLANNER it is possible to make deductions and to achieve goals. Goals are achieved by building a tree of goals and subgoals terminating in basic actions, and if a plan fails to achieve its goal the system can look for another one. The conversations produced by the program and its user are structured entirely by the user: the program

never takes the initiative, and it treats the utterances of the user as independent promptings (i.e. it never uses one utterance to form an expectation about the next). For our present purposes, the chief interest of the system lies in the way it represents meaning and the way it achieves goals.

1.5.2 Speech Acts

Problems of human conversation have recently interested philosophers, notably Austin (1962) and Searle (1969), and since they put a number of issues very clearly it is worth reviewing their main ideas. Most work in the philosophy of language has been concerned with problems of meaning and truth, a typical problem being "How is it that some strings of words make sense while others make nonsense". This kind of investigation tends to think of sentences as neutral objects which either do or do not express a meaning, rather than as things which people use for particular purposes. In his book "How to do things with words", Austin (1962) examines language from the second, less common point of view.

Austin distinguishes three kinds of "speech act": the locutionary act, the illocutionary act, and the perlocutionary act. The best way to make the distinction is to consider an example, and we will use the one word

command "Run!" for this purpose. In the last sentence, we used this command in a purely locutionary way, as we only quoted it: we had no intention of causing the reader to run away, and we assume he has stayed put. For the command to be an illocutionary act, the speaker would have to "really mean it", and thus intend his hearer to obey it. On reporting this event, the hearer might say "he told me to run". There is no implication here that the hearer actually realised the speaker's intention; but if he did, the command would become a perlocutionary act. If "Run!" were a perlocutionary act, the hearer (and runner) would report it by saying "he persuaded me to run", implying "he meant me to, and I did".

Normal conversation consists almost entirely of illocutionary acts, some of which are also perlocutionary, and Austin and Searle are chiefly interested in examining the phenomenon of illocution. In other words, they want to define more exactly what it is to say something and really mean it. What, for example, distinguishes a real question from a quoted one? Searle gives this answer: for a speaker to make an illocutionary act rather than just a locutionary one, he must accept that his act is subject to certain rules. Given a question Q uttered by speaker S to hearer H, the rules would include the following (adapted from Searle, 1969):

(a) Content: Q should be a proposition or propositional function.

(b) Preparatory: S should not know the answer, and it should not be obvious to S and H that H was about to provide the information anyway. (In short, Q should be necessary).

(c) Sincerity: S should want to know the answer to Q.

(d) Essential: Q counts as an attempt to elicit information from H.

Thus a speaker who is really asking a question should accept that a complaint about his sincerity (say) is relevant, just as a tennis player who is really serving (as opposed to knocking up) should accept that his action be judged by whether the ball lands in the correct court (to use an analogy of Alston's (1964)).

The above ideas are relevant in two ways: first, they provide a way of thinking about the language perception process; and second, they help us to define the range of appropriate responses to a given remark. It is interesting to think of language perception as occurring in three stages: first, deciding the locutionary force of the utterance, then its illocutionary force, then its perlocutionary force. The first stage is that of finding out the content of the utterance; having done this, the perceiver has to decide whether it is a real question, or real command, or whatever;

finally, he has to decide on his reaction to it, or lack of reaction, and thus to determine its perlocutionary force. The range of appropriate replies reflects this threefold division. Consider the following replies to the question "What is your father called?"

1. Did you say "father" or "brother"?
2. What does "father" mean?
3. I just told you.
4. I was just about to tell you.
5. You're not really interested in my father.
6. I'm not telling you.
7. His name is Albert.

Of these seven replies, 1 and 2 are generated during the locutionary stage of analysis; 3, 4 and 5 during the illocutionary stage; and 6 and 7 during the perlocutionary stage. In 3, for instance, the content of the utterance is understood but it is not accepted as a real question as it breaks rule (b) above; and in 6, the utterance is accepted as a genuine question but an answer is refused. It will be seen later that our program ignores these distinctions, and this must be judged one of its main weaknesses: for example, it has no representation of the rules for illocutionary acts, and is thus not at all put out if its companion asks it the same question repeatedly,

or asks it a question which there can be no possible reason for asking.

2. DESCRIPTION OF THE PROGRAM

2.1 Introduction

The aim of this section is to explain how the program works; we are not yet concerned with discussing it or comparing it with other work. For the sake of clarity, we will disect the mind of John rather than that of a generalised robot, though it should be remembered that Mary is virtually an identical copy. The actual program is written so that every variable begins with a Z, but when the program is used it is compiled twice, substituting J for Z the first time and M for Z the second. Thus there are two matching sets of variables and function definitions representing the two minds. Before running the program, some of the variables in each half are initialised, and it is here that the only differences between the two minds are introduced. To take two examples, the variables holding a robot's name are given different values, and so are the variables holding a robot's goal. Thus John starts off knowing that his name is "John" and his goal is (say) to get in, while Mary knows that her name is "Mary", and might have a different goal, or none at all.

In describing a program of this kind, one feels compelled to use mentalistic terms such as memory, perception, knowledge, to refer to its various structures and processes. These concepts are essential to the understanding of the

program; if we invented new words to replace them, then the reader could only understand what was going on by secretly translating each new word into its mentalistic equivalent. All the same, it is worth discussing briefly what we mean when we say that a given structure is, for example, a "memory", or that a given process is "perception".

To take a very simple case, there is a variable in John's mind called JKYOU which is given the value "Mary" at the start. We want to say that this is John's knowledge of the name of his partner. The reason we say this is that if the variable is set differently - to "Fred", say - then when John wants to attract his partner's attention he calls out "Fred!" instead of "Mary!". In other words, we decide which mentalistic term to use by examining the kinds of behaviour which result from the structure in question being in different states. Of course, the variable JKYOU only affects behaviour as it does because the rest of the program reacts with it in a particular way: if the variable was hidden away and never consulted, then it would no longer be John's knowledge of his partner's name. So the variable is not a piece of knowledge by virtue of its intrinsic structure, but because it is used in a particular way by the rest of the program.

In describing the various structures of the program, then, we will use mentalistic terms freely, but will also

give some indication of why these terms are justified, by showing how the structures influence behaviour and how they interact with the rest of the program. The most difficult problem here is to describe how one bit of the program interacts with the other bits without having first described all the other bits. There is no adequate way round this problem; the best we can do is to describe the interaction in general terms and hope that the details will fall into place later.

Before describing the program section by section, it may help if we indicate how the minds and bodies of the robots are represented inside the computer.

The program has three main parts, one representing John's mind, one Mary's mind, and the other the physical world. The third part contains the bodies of the robots, the door, the bolt, and the laws of nature. All the variable names in John's mind begin with the letter J, those in Mary's mind begin with M, and those in the physical world begin with W. Thus the current states of John's mind, Mary's mind, and the world, are determined by the values of the variables beginning with J, M and W respectively.

Different parts of a program communicate by changing the values of each other's variables, and the process by

which this is done is called "assignment". If we have two variables V1 and V2, then the instruction V1 —> V2 gives V2 the same value as V1, leaving V1 unchanged. So if V1 and V2 are originally 3 and 5, the instruction V1 —> V2 makes them both 3. And if JVAR is a variable in John's mind, and WVAR a variable in the physical world, JVAR —> WVAR causes John to alter the state of the world - that is, it is either an action or an utterance; and WVAR —> JVAR is correspondingly a change in John's mind induced by the world, or a perception.

The program is written so that direct communication of this kind only occurs between a robot and the world, never between the robots. The instruction JVAR —> MVAR would imply that John's mind could influence Mary's mind directly; in other words it would be telepathy, and we assume it to be impossible. If John wishes to communicate with Mary he must go via the world, using an action or an utterance. This would require two instructions: JVAR —> WVAR followed by WVAR —> MVAR.

The variables actually used to hold the positions of objects in the world and the last utterance made by a robot are WOBJECTS and WMESSAGE. Bearing this in mind, we can sum up the various interactions between parts of the program as follows:

JVAR —> WMESSAGE - John speaking

WMESSAGE —> MVAR - Mary hearing what he said

JVAR —> WOBJECTS - John performing an action

WOBJECTS —> JVAR - John perceiving the object positions

JVAR1 —> JVAR2 - John thinking

JVAR —> MVAR - Telepathy, and thus impossible

The instruction WVAR1 —>WVAR2 would indicate a change in the world not caused by either robot: this also never happens. As far as the robots are concerned, their environment is passive, and they only inspect it for possible changes when one of them has carried out an action.

One of the main problems in setting up a conversational system inside a computer is that computers are serial systems: they can only do one thing at a time. So it is impossible to have John and Mary thinking at the same time, as would occur in real life. There are two ways round this problem, one of which is good and difficult, the other not so good and easy, and we have chosen the latter. The most realistic method is to use a time-sharing system so that the minds work in pseudo-parallel. The way we actually do it is to have a short "chairman" procedure which calls the robots alternately. The chairman begins by arousing John, and John thinks until either he wants to speak, or he wants to swap control to Mary for some other reason, or he has nothing more to do. Then control returns to the chairman and Mary is aroused, and so on. The drawback to this method is that a robot can never say something

which interrupts the other's thought processes, but we use it all the same because it is so much easier to implement.

To start John thinking, the chairman calls a function JEAROUSE; Mary is correspondingly activated by MEAROUSE. When John makes an utterance and control returns to the chairman, the place John has reached in his thoughts is saved, so that next time he is aroused he carries on from where he left off rather than having to relive all of his previous mental experience.

In describing the program, we will have to refer a lot to association lists, since these are the main data-structures that we use to store information. An association list is a list of pairs of items, the first of each pair being a constant, and the second being allowed to vary. The list which stores the state of the world, for example, might be [JOHN IN MARY OUT DOOR SHUT BOLT UP], and if John pushes the door open and goes out it would change to [JOHN OUT MARY OUT DOOR OPEN BOLT UP]: only the second item in each pair varies. If we want to indicate the general structure of such a list, we will use variables of the form X1, X2, X3 etc. to refer to the items which can vary, e.g. [JOHN X1 MARY X2 DOOR X3 BOLT X4], and it must be remembered that these variables are not in the program, but are only used for purposes of exposition.

Finally, a few technical details. The program is written in POP-2, and requires 65 blocks of store to compile.

It prints out conversations at the rate of an utterance every two or three seconds. In describing the program, we will try to avoid referring too much to the programming language, but some reference to it is inevitable. The program relies mainly on non-numerical computations involving lists, and for an introduction to this kind of programming the reader is referred to Fox (1966). A full description of POP-2 is given by Burstall et.al.,(1972).

2.2 Perception and Action

Since perception and action are the means by which John interacts with the world, we begin with a description of the world. This consists of two variables, WOBJECTS and WMESSAGE, and three functions, WMOVE, WPUSH and WSLIDE. The variables hold the current state of the world, and the functions specify the consequences of the three possible actions.

WOBJECTS is a list with the following structure: [JOHN X1 MARY X1 BOLT X2 DOOR X3] X1 can be either IN or OUT, X2 either UP or DOWN, and X3 OPEN or SHUT. There are only four objects in the world, each of which can only be in one of two positions, so the above list provides a complete description once the variables are filled in. When the program starts, WOBJECTS might be [JOHN OUT MARY IN BOLT DOWN DOOR SHUT].

WMESSAGE is also a list, and holds the last utterance

made by a robot. For example, it might be [MAY I ASK YOU SOMETHING]. When a robot "reads" the message he resets WMESSAGE TO [ ], the empty list, and unless he says something in reply, WMESSAGE remains empty.

The functions all take one argument, the name of the robot carrying out the action. So if John wants to move, he must call WMOVE (JOHN), and if Mary wants to slide the bolt she must call WSLIDE (MARY). For John to call one of these functions with MARY as argument, or vice versa, is not allowed. An example of one of the function bodies is given below, translated into English. It says that a robot can only move through the door if the door is open:

WMOVE (ROBOT)
if the symbol after DOOR in WOBJECTS is OPEN
then if the symbol after ROBOT (i.e. JOHN or MARY)
in WOBJECTS is IN, change it to OUT
else if it is OUT, change it to IN.
(If the symbol after DOOR is SHUT, do nothing).

Thus if WOBJECTS is [JOHN OUT MARY IN DOOR OPEN BOLT UP] the call of WMOVE (JOHN) will change the JOHN OUT to JOHN IN, while WMOVE (MARY) will change MARY IN to MARY OUT. But if WOBJECTS is [JOHN OUT MARY IN DOOR SHUT BOLT UP] neither instruction will cause any alteration.

Similarly, WSLIDE alters the position of the bolt provided that the robot carrying out the action is IN,

while WPUSH alters the position of the door provided that the bolt is UP.

It should be evident from the above account that there are four possible object positions and three possible actions; these limits arise from the nature of the world. It is possible to further limit the robots, and in different ways. For example, we can arrange that John can only see himself and the bolt, and can only move or push the door, while Mary might have different limits. The program is in fact written so that John only sees and acts according to what he believes he can do: in other words, if he believes he cannot slide the bolt, then he never tries to, and the belief causes his limitation. There are two lists which hold these beliefs, JKSEE and JKACTS; the former is a list of those objects John can see, and the latter is a list of the actions he can perform. In the case mentioned above, JKSEE would be [JOHN BOLT] and JKACTS [MOVE PUSH].

Perception is basically an updating of John's memory resulting from an examination of those objects he can see. If an object changes position, John's memory is not updated automatically; he has to notice the change. The function JKLOOK is used to take note of the current object positions. It takes the list JKSEE and finds the position of every object on it, entering each position in John's memory. If an object is not in JKSEE, John retains his

old belief about it: for instance, if Mary has told him that she is out, and he cannot see her, he continues to accept this as being her position until she tells him otherwise.

An action is always part of a plan to achieve a goal. If John cannot perform an action (e.g. if he can't slide the bolt) then he will never accept a plan which says that he is to do so. Carrying out an action is simply a matter of calling the appropriate function (WMOVE, WSLIDE or WPUSH) with JOHN as argument.

We have said that JKSEE and JKACTS represent both John's actual limitations, and his beliefs about those limitations. They reveal themselves as beliefs in the following circumstances:

(a) If Mary asks John whether he can perform a particular action, he finds out whether the action is in JKACTS and says "Yes" if so and "No" if not.

(b) If Mary suggests a plan which assigns to John an action not in JKACTS he tells her that he can't do the action.

(c) If Mary tells John that the bolt is up (say), while he believes the opposite, he contradicts her if BOLT is in JKSEE but accepts her word for it if not. In the first case, he says "I disagree" and leaves his memory unaltered; in the second case he says "I see" and updates his memory accordingly.

2.3 Memory

2.3.1 Pre-amble

The term "memory" can cover a wide area, and we will be using it here in a rather special sense. To make this sense clear, we begin by making two distinctions between different kinds of knowledge.

The first distinction is between knowledge held in global variables and knowledge held in local variables. A global variable is one which exists independently of the particular procedure now running. It is used to hold knowledge which will be used generally throughout the system, and it will be consulted by a number of different procedures. Local variables are tied to a particular procedure, and die as soon as that procedure ceases to run. They are used to store information which is not required by any other procedure. To some extent one can carry over the distinction into everyday life, though it is not so sharp. Most telephone numbers are examples of things we remember only so long as we need them for a particular purpose, though there are some numbers, such as 999, which are part of our general, "global" knowledge.

The second distinction is between constant and variable knowledge. A lot of John's knowledge - most of it in fact - never changes during a run of the program. His knowledge about how to make plans, how to use language, what he can see and do, how many objects there are in the world, his

name, his partner's name: all this remains constant. It can sometimes be altered by the operator before a run, but never by the program during a run. The kinds of knowledge which vary during a run are these: his knowledge of object positions, his theory of the consequences of actions, his model of Mary's mind, his current plan and current goal, his last utterance and the last utterance of his partner ... and so on.

We will use the term "memory" to refer to John's global and variable knowledge, in the above senses. In other words, we are talking about that knowledge which is generally applicable and which changes as a result of experience. This can be subdivided into two main kinds: John's model of the world, and his model of the mind of his partner, Mary.

2.3.2 Model of World

John's world model is held in two variables, JKWORLD and JKRULES. JKWORLD holds John's beliefs about the positions of objects - i.e. the current state of the world; and JKRULES holds his theory of how the world works - his beliefs about the consequences of actions.

Positions of Objects

The model of object positions (JKWORLD) is an association list with the structure [JOHN X1 MARY X1 BOLT X2 DOOR X3]. The significant values of X1, X2 and X3

are as follows:

X1: IN, OUT, UNDEF

X2: UP, DOWN, UNDEF

X3: OPEN, SHUT, UNDEF.

The only unfamiliar term here is UNDEF; this is used throughout the program to mean "I don't know". So the model has a double function: it records whether John knows the object positions and, if so, what he thinks they are.

At the start of a run of the program, JKWORLD is set to [JOHN UNDEF MARY UNDEF BOLT UNDEF DOOR UNDEF]. When John takes a look at the world, he will find the positions of those objects he can see and fill them in: e.g. [JOHN OUT MARY UNDEF BOLT UP DOOR UNDEF]. The ability to set up the original list, and the knowledge of which positions can be associated with which objects, might be said to constitute John's knowledge of the logical possibilities in his world, and it exists prior to his experiences on a given run. There are no facilities for changing his ideas as to what is logically possible (e.g. by allowing for the discovery of new objects or intermediate positions).

We now take an example from this list, the symbol DOOR and its "partner" (UNDEF, OPEN or SHUT) and try to show why this can be taken as John's belief as to the position of the door. The following kinds of behaviour

are involved:

(a) If John is asked "Is the door open" he will answer "Yes", "No", or "I don't know" depending on whether the symbol after DOOR in JKWORLD is OPEN , SHUT or UNDEF.

(b) If John wants to change his position by moving through the door, and knows that it is necessary to get the door open first, then if the symbol after DOOR is OPEN, he will suggest a plan involving only his moving; if it is SHUT, he will suggest that the goal of getting it OPEN is achieved first; and if it is UNDEF, he will ask Mary whether it is open before suggesting one plan or the other.

(c) If John's original goal is to get the door open, then if the symbol after DOOR is OPEN he will keep returning control to the chairman without saying anything, but if it is SHUT he will try to get it open and will ask Mary for help if he gets stuck.

So the list JKWORLD influences John's replies to questions, his plan formation, and his satisfaction with the current state of affairs, and in each case it acts as if it constituted his beliefs about the positions of the objects.

Consequences of Events

John's theory of how the world works consists of three rules, held in JKRULES. To simplify the problem, we have made very strong assumptions about the form of a rule, and John can thus only apprehend a limited number. A rule in

His theory is based on three parts: an event, a situation and a consequence. The rule says, in effect, "this event will lead to this consequence provided that the situation is this". For example: "if a robot pushes the door, the door will change position provided that the bolt is up".

An event can be one of three possibilities: a robot pushing the door, or sliding the bolt, or moving. In other words, there will be rules to predict the consequences of all three possible actions. These events are represented in the program by lists of the form [ROBOT X1] where X1 can be MOVE, SLIDE or PUSH.

A situation specifies the state of one of the objects; e.g. in the example above the situation was "the bolt is up". This is a severe limitation since it assumes that only one of the objects will ever be a relevant influence. In John's world this happens to be the case, but in most worlds one would need more complicated rules. The only other possible specification of the situation is the word ANY, which indicates that the situation is irrelevant. So one can have the rule: "If you push the door, the door changes position, regardless of the positions of other objects". The actual representation of the situation is [BOLT UP] if the proviso is that the bolt be up, and [ANY] if there is no proviso.

The consequence of the event can be one of three

things: either an object changes position, or nothing happens, or the consequence is unknown. If the object in question is the door, for example, and the event is [ROBOT PUSH], then the rule means that pushing will open the door if it is shut, and shut it if it is open. Again this is a simplification which relies on a special feature of this particular world, namely the fact that actions have symmetrical effects. The action which brings about a change is the same as the one which can reverse it. So if an action is performed twice in succession, the original state of the world is always maintained. In the program, the consequences mentioned above are represented by the lists [X2], [NOTHING] and [UNDEF] respectively, where X2 can be ROBOT, DOOR or BOLT.

The syntax for a complete rule is as follows: [EVT X3 SIT X4 RES X5]. The words EVT, SIT and RES are short for Event, Situation and Result (i.e. consequence) and X3, X4 and X5 are lists of the kinds referred to above. To make this clearer we now give some examples of rules together with their English equivalents.

[EVT [ROBOT PUSH] SIT [ANY] RES [NOTHING]]

"If you push the door, nothing happens"

[EVT [ROBOT MOVE] SIT [DOOR OPEN] RES [ROBOT]]

"If you move when the door is open, you change position"

[EVT [ROBOT SLIDE] SIT [ANY] RES [BOLT]]

"If you slide the bolt, it changes position"

[EVT [ROBOT SLIDE] SIT [ANY] RES[UNDEF]]

"I don't know what happens if you slide the bolt"

The list of rules, JKRULES, consists of three rules only, one for each event. The purpose of the simplifications mentioned above is to keep this list short and easy to manage, since the issue of how we make rules of this kind is not our main concern. So JKRULES is always a list of three lists, each sub-list representing a rule, there being one rule each for the actions MOVE, SLIDE and PUSH.

John's theory of how the world works influences his behaviour in two ways: first, it affects the plans he makes to achieve his goals; and second, it affects his response to a plan suggested by Mary.

Suppose John is in the following state as regards his immediate goal, his model of object positions, and his theory:

Goal: [DOOR OPEN]
JKWORLD: [JOHN OUT MARY IN DOOR SHUT BOLT DOWN]
JKRULES: [[EVT [ROBOT PUSH] SIT [BOLT UP] RES [DOOR]]
[EVT [ROBOT SLIDE] SIT [ANY] RES [NOTHING]]
[EVT[ROBOT MOVE] SIT [ANY] RES [NOTHING]]]

If he is supposed to suggest a plan, John will first make sure that the goal is not achieved. Then he looks through

the list of rules until he finds a rule whose RES matches the goal - i.e. a rule which shows him how to alter the position of the object mentioned in the goal. Such a rule may not exist, but in this case the first rule meets his requirements since its RES is [DOOR]. John next examines the EVT of the rule, and decides who is to carry out the action. If he can do it, he fills in JOHN for ROBOT throughout the rule; if he cannot but Mary can he fills in MARY. This is called making the rule specific. It now reads: [EVT [JOHN PUSH] SIT [BOLT UP] RES [DOOR]]. Next, John examines the SIT, and finds out whether it is achieved. If not, he adds it as a preliminary subgoal. Thus in this case he suggests the plan: 1. We get the bolt up 2. I push the door. If the bolt was already up, he would have left out the first of these subgoals.

If Mary has suggested a plan, judging its adequacy is a rather simpler matter, but makes use of the rules in a similar way. Suppose that in the above circumstances Mary suggested that she should push the door. Since the event involves the action PUSH, the rule containing the EVT [ROBOT PUSH] will be chosen as the relevant one; then the result of carrying out the action will be computed using this rule. Since the actual situation is [BOLT DOWN] and the necessary situation is [BOLT UP], John concludes that the plan will not alter the position of the door, and

explains to Mary the rule on which he bases his rejection. In fact this rule is transformed to its equivalent form; [EVT [ROBOT PUSH] SIT [BOLT DOWN] RES [NOTHING]], and John says "If you push the door when the bolt is down, nothing happens". So the list of rules is used (a) to evaluate the plan, and (b) to give his reason for rejecting it if he finds it inadequate. It should be evident that with different rules John will make different plans, give different evaluations, and give different reasons if his evaluations are negative.

To complete our discussion of John's theory, we consider how his rules are changed as a result of experience. Whenever an action has taken place, John assesses its result, and alters his rules if their prediction was incorrect. There is no point in going into exact detail here, but we will give three illustrations.

(a) Suppose that John's rule for the action MOVE is [RES [ROBOT MOVE] SIT [ANY] RES[UNDEF]] - that is, he doesn't know what to expect. Then he will simply examine the world for any changes, and replace [UNDEF] with the name of the kind of object which moved, or with [NOTHING] if none did. So if Mary tried to move, and changed her position from IN to OUT, John will make his rule

[RES [ROBOT MOVE] SIT [ANY] RES [ROBOT]]

- in other words, he will assume that the action always has

the observed effect.

(b) Suppose that the next time someone moved, nothing happened. This contradicts the rule that if a robot moves, he changes position, and in such cases John looks for some proviso not achieved in his present environment. He will make an arbitrary choice from those objects which are not already mentioned in the rule: that is, from the bolt or the door. If he selects the bolt, he decides that the reason why nothing happened this time was that the bolt was down, and changes his rule to:

[EVT [ROBOT MOVE] SIT [BOLT UP] RES [ROBOT]]

(c) Having got the bolt up, (but not opened the door), John tries to move again and still nothing happens. So again he has to alter his rule. He looks for an object position not mentioned in the present rule, and since ROBOT and BOLT are mentioned, only DOOR is left. Since the door is now shut, he decides that robots only get places by moving when the door is open, and his final rule, the correct one, is

[EVT [ROBOT MOVE] SIT [DOOR OPEN] RES [ROBOT]].

2.3.3 Model of Partner

John's model of Mary consists of the following variables:

(a) JKXSEE - What Mary can see

(b) JKXACTS - What Mary can do

(c) JKXRULES - Mary's beliefs about how the world works.

We now consider these in turn.

Model of what Mary can see - JKXSEE

JKXSEE has the structure [JOHN X1 MARY X1 BOLT X1 DOOR X1] where X1 can be 1, ∅ or UNDEF. So a possible value of JKXSEE is [JOHN 1 MARY 1 BOLT UNDEF DOOR ∅] which indicates that Mary can see John and herself, but not the door, and that John doesn't know whether she can see the bolt.

To show how this list affects behaviour, we consider a possible case. Suppose that John, for some reason, wants to know whether the bolt is up, and his models of what Mary can see and of the positions of the objects are in this state:

JKXSEE : [JOHN 1 MARY 1 BOLT UNDEF DOOR ∅]

JKWORLD : [JOHN OUT MARY IN BOLT UNDEF DOOR SHUT]

John will first look in JKWORLD, and find that he doesn't know the position of the bolt. Then he will see whether Mary knows it, by examining JKXSEE. If she doesn't, he won't try asking but will give up trying to find out. If (as in this case) she either knows or might know, he asks. So John will ask the question "Is the bolt up".

In reply Mary will either answer "Yes", "No", or "I don't know". If the answer is "Yes" or "No", John updates JKWORLD accordingly and puts a 1 after BOLT in JKXSEE. If Mary replies "I don't know", he puts ∅ after BOLT in JKXSEE and leaves JKWORLD as it is. Next time he wants to know

about the bolt, he finds that Mary doesn't know about it (can't see it) and doesn't bother to ask. It is this device which stops him from asking again and again when he is in a situation where he needs to know something which Mary doesn't know either. The original question leaves his model of object positions unaltered, but changes his model of what Mary knows, and thus he avoids going into a loop.

Model of what Mary can do - JKXACTS

The next variable, JKXACTS, has the structure [PUSH X1 MOVE X1 SLIDE X1] where again X1 can be 1, Ø or UNDEF. So if JKXACTS is [PUSH 1 MOVE Ø SLIDE UNDEF], John knows that Mary can push the door and cannot move, but doesn't know whether or not she can slide the bolt. Some examples of how this list is used are now given.

(a) If John is working out a plan which requires that Mary do something, then he makes sure she can do it before suggesting the plan. If he knows she cannot, he gives up the plan; if he knows she can, he goes ahead and suggests it.

(b) If John doesn't know whether Mary can (say) slide the bolt, and needs to know, he asks her ("Can you slide the bolt"). She must know the answer to this and says "Yes" or "No" (after consulting MKACTS - see 2.3.2). John enters 1 or Ø after SLIDE in JKXACTS and proceeds with his routine to make a plan.

(c) There are two other ways in which John could find out whether Mary can do something. If she tells him she has performed an action, he assumes that this means she can do it; and if she tells him she can't do an action which he suggested she did, he also updates JKXACTS accordingly. This latter situation would only arise if JKXACTS was wrongly set at the beginning of a run - if, for example, the operator put a 1 after MOVE when in fact Mary was unable to move. John might then make a remark such as "I suggest we get the door open and then you move", to which Mary would respond by telling him that she couldn't move. John would take her word for it, and suggest a different plan next time.

Model of Mary's beliefs as to the consequences of events - JKXRULES

JKXRULES is a list of those rules of Mary's that John believes he knows; thus it can contain up to three rules. The rules are expressed in the same format as is used in JKRULES. Two examples of the uses of JKXRULES are now given:

(a) Suppose Mary has suggested a plan which John believes to be no good (i.e. his rules say that the plan won't achieve its goal). If he doesn't know what Mary's relevant rule is, he gives the rule by which he rejected the plan, and this leads to a discussion during which he will discover her rule and might change it. If, on the other hand, he

knows her rule, he then applies a test to decide whose rule is "better".

A rule is considered to be better than a rival if it is more complex - i.e. if it contains a proviso not mentioned in the rival. So of the rules:

(i) If you push the door, the door opens

(ii) If you push the door, nothing happens

(iii) If you push the door when the bolt is down, the door changes position

(iv) If you push the door when the bolt is up, the door changes position,

(iii) and (iv) are better than (i) and (ii), (i) and (ii) are equally good, and (iii) and (iv) are equally good.

If John's rule is better than the rule he ascribes to Mary, he counters her plan suggestion by explaining the rule by which he rejects it. If his rule is not better, he agrees to try the plan even though he believes it won't work. This means that even when they believe different rules, with no obvious reason for choosing one rule rather than the other, they can agree on some plan; and when the plan has been carried out, somebody will be proved wrong and will change his rule.

(b) If John has just finished assessing the consequences of an action, and he finds that his rule predicted the consequence correctly, while the one he ascribes to Mary failed, he tells Mary he wants to explain something, and

then gives her his more successful rule.

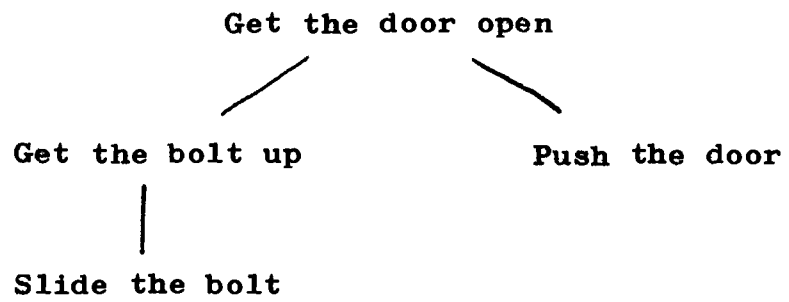
2.4 Plans

2.4.1 Pre-amble

In the last analysis, the minds of men, mice and mosquitos exist for one purpose only: to churn out effective sequences of actions. When we observe insects and other lower animals setting about this task, we cannot but admire the cleverness which is implicit in what they do. Once we understand what they are doing, we recognise that it is both purposeful and reasonable. But at the same time, we see that the animal itself is completely ignorant of the reasons why its actions work so well, and we claim that although we may understand what it is doing, the animal itself is unable to. If we alter the animal's normal habitat so that its customary procedures cease to work, it usually fails to adapt to the new situation, and carries on in the old way, now hopelessly inappropriate.

If we are to make robots which can discuss what they are doing and react intelligently to a changed situation, we will have to give them what human beings (to some extent) have, and insects lack: the ability to understand what they are doing. So the question to be considered is this: what kinds of knowledge are involved in understanding one's own behaviour, and how can this knowledge be represented?

Suppose that John is working alone, that his goal is to get the door open, and that the world situation is: [JOHN IN MARY OUT BOLT DOWN DOOR SHUT]. The behaviour necessary to achieve the goal is 1. Slide the bolt up, and 2. Push the door open. If John is to understand why he performs these actions, the first requirement is that he knows what each is meant to achieve, and can recognise whether or not it has succeeded in its object. In other words, he must be able to assign the following description to his actions



and to recognise when the bolt is up or the door is open. If sliding the bolt fails to get it up, he must look for some other way of doing so, rather than going ahead with pushing the door.

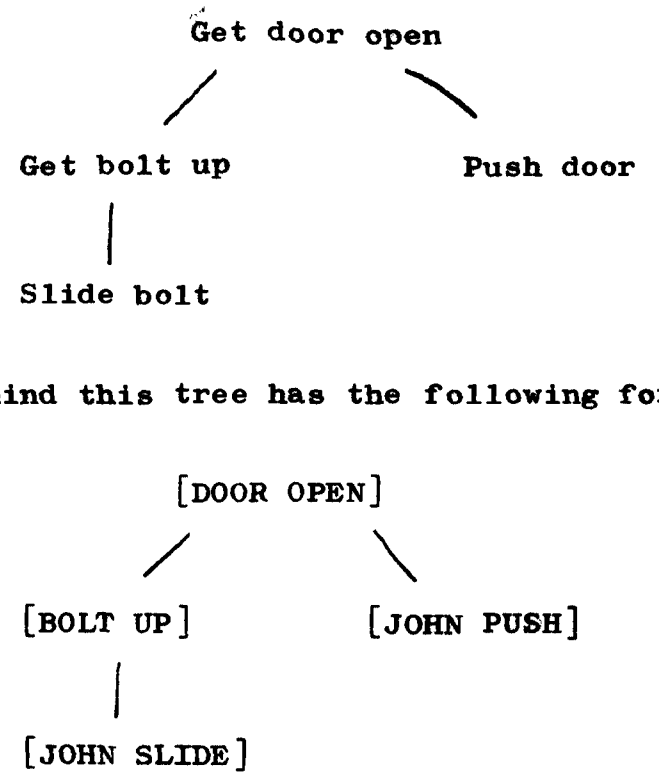
If the first requirement of understanding is knowing that (for example) sliding the bolt is meant to get the bolt up, the second requirement is knowing <u>why</u> the action is invoked as a plan to achieve the goal. To display this level of understanding John must be able to work out the consequences of any given plan in a given situation, and

thus to judge whether or not it will work. What is more, if a plan turns out differently from what he expected, he must alter his beliefs about the world so that they account for this new occurrence.

To sum this up: If John is to understand an action, we demand that he knows (i) what goal the action is supposed to achieve, and (ii) why the action is expected to achieve the goal. In the next two sections we discuss (i) how John represents the goal structure which underlies his actions, and how this structure is built and used; and (ii) how John makes a suitable plan to achieve a goal, or justifies his plan, or evaluates someone else's plan, or justifies his evaluation of someone else's plan.

2.4.2 How plans are represented and used

In the last section, we saw that the relation between a sequence of actions and the main goal they were meant to achieve could be represented by a tree in which the main goal was the root and the actions were the terminal nodes. We reproduce the tree below, since we are going to continue to use it as an example. It will be recalled that, at the start, John is in, Mary out, the bolt is down and the door shut. The goal is to get the door open, and John is working on his own. Mary will play no part in the example.



In John's mind this tree has the following form:

There are two kinds of goal on this tree. [DOOR OPEN] and [BOLT UP] are both called "situations" (or states-of-affairs); and [JOHN PUSH] and [JOHN SLIDE] are "events". Situations consist of an object followed by a possible position for it; events consist of a robot followed by an action. The terminal nodes of a tree will always be events, and the other nodes will be situations.

It is convenient to represent situations this way because we can use a representation such as [DOOR OPEN] to find out whether this situation has in fact been brought about. It is only necessary to find which word is associated with DOOR in JKWORLD and to compare this word with OPEN. Similarly, [JOHN SLIDE] contains the information

we need in order to determine whether the event is possible: by looking in JKACTS we find out whether or not John can slide the bolt. Had the event been [MARY SLIDE] we would have examined JKXACTS. In short, we have represented events and situations so that it is easy to find out whether the events are possible and whether or not the situations are brought about.

The tree as we have described it up to now has goals at each node, the goals being either situations or events. The tree John uses is more complicated, as it can have three other items of information attached to each node. We call these the state of the goal, its actor, and its plan, and they will now be explained in turn.

State

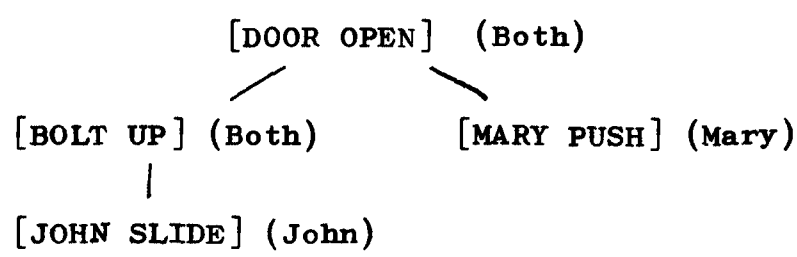
It would seem at first that a goal can have only two states: either it is done or not done; but a threefold classification is in fact more useful. The reason is that we want to distinguish goals which are not yet done but might be from those which have been given up as impossible. So we classify goals as ACHIEVED, FAILED, or NOTYET. Most goals on the tree will be marked NOTYET, since goals which have been achieved or failed will be removed when the tree is periodically revised. How this happens will be demonstrated later.

If a goal is done, it is marked ACHIEVED. A goal is marked FAILED if John is unable to think of a plan that has

a chance of achieving it. All goals which are still being worked on, and are neither achieved nor abandoned as impossible, are marked NOTYET.

Actor

A goal's "actor" is the robot or robots responsible for achieving it. The actor is JOHN, MARY or BOTH, depending on whether one of the robots is responsible or they are jointly responsible. If John is trying to achieve a goal on his own, then the goal and all its subgoals will be his responsibility, and he will not consult Mary when choosing his plans. But if the goal is a joint one, the plan will have to be agreed, and it may involve actions by both of them. Thus if John and Mary were trying to achieve the goal [DOOR OPEN] together, and only Mary could push the door, the planning tree complete with actors would look like this:



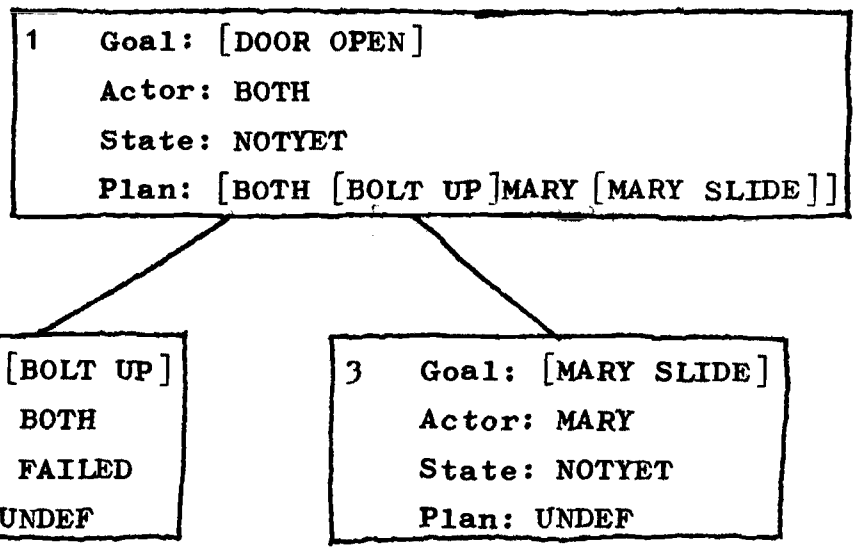
The point is that if a plan is to be carried out jointly, the parties must know both what has to be done and who has to do it. The decision as to who carries out each part of the plan must be specified when the plan is agreed on. In the examples that occur in our simple setting, it is

always obvious who has responsibility: a robot is responsible for his own action, and situation goals are the responsibility of both. But in normal circumstances it is quite common for one party to be assigned a task such as "Clear the table" and to be left to make the subgoal choices himself.

Plan

The third attachment to a goal on the tree is the plan that has been tried in order to achieve it, if such a plan exists. When a goal is first put on the tree there is no plan associated with it, and John simply chooses one by the normal method. When he does so, the plan is not only added to the tree as subgoals, but is saved by being attached to the node for which it was made. If the plan fails, the subgoals which it comprises will be wiped off the tree and control returns to the original goal. It is vital that the same plan is not tried again. If John has changed his theory of the world as a result of the failure of the plan, then his next plan will be different; but he may not have changed his theory, and the plan he selects may thus be the same. This is why it is necessary to keep a record of the old plan. Whenever a new plan is generated at a node, it is compared with the old one, and if it is the same, the node in question is marked FAILED. In other words, if you can't think of a plan to achieve a goal other than the plan which has just failed, you fail the goal.

The complete tree, then, has four items attached to each node: the goal, state, actor, and plan. The representation of the tree in the program is not too important, but we mention it briefly. There are five variables, JPTREE, JPGOAL, JPACTOR, JPSTATE and JPPLAN, all of which are actually global, but which are really used as if they were local variables of procedures called ROUTINES which operate the planning tree. (This paradoxical remark will be cleared up later when we consider routines). Every node is given an index number, and the tree as a whole is hung from an empty node with the index $\emptyset$. (This is for programming convenience.) The five variables mentioned above all hold association lists. JPTREE associates node indexes with the indexes of their subgoals, and JPGOAL, JPACTOR etc. associate node indexes with goals, actors, states and plans. We give below a complete tree and its representation in the lists.



JPTREE: [ø [1] 1 [2 3]]

JPGOAL: [1 [DOOR OPEN] 2 [BOLT UP] 3 [MARY SLIDE]]

JPACTOR: [1 BOTH 2 BOTH 3 MARY]

JPSTATE: [1 NOTYET 2 FAILED 3 NOTYET]

JPPLAN: [1 [BOTH [BOLT UP] MARY [MARY SLIDE]] 2 UNDEF 3 UNDEF]

We now give a full-scale example to show how the planning tree is built and how it is used to control behaviour. To begin with, it is necessary to define the starting position by giving the initial values of the key variables. We will assume that John can see all the objects and perform all the actions.

Goal (JKGOAL): [DOOR OPEN]

Model of objects (JKWORLD): [JOHN IN MARY OUT DOOR SHUT BOLT DOWN]

Theory (JKRULES): [[EVT [ROBOT PUSH] SIT [ANY] RES [DOOR]]

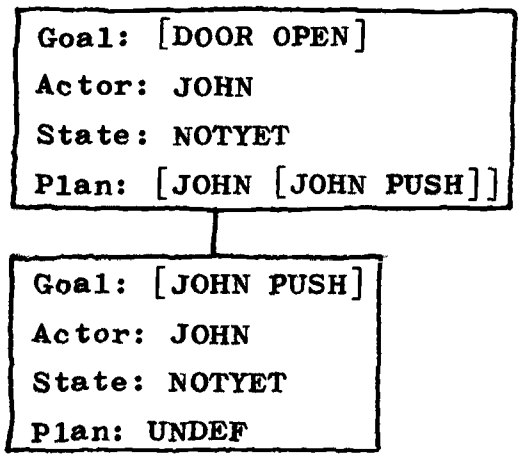
[EVT [ROBOT SLIDE] SIT [ANY] RES [BOLT]]

[EVT [ROBOT MOVE] SIT [ANY] RES [UNDEF]]]

The first thing that John does is to construct the top node of the tree, the one which represents his main goal. Since he is attempting the goal on his own it is his responsibility, and since the door is now shut the goal is not yet achieved. So the node he constructs is:

Goal: [DOOR OPEN]
Actor: JOHN
State: NOTYET
Plan: UNDEF

This goal is a situation and not an event. If it were an event, he would now carry out the action it specified, but as it is not, he makes a plan to achieve it and attaches the plan to the tree. He believes that if you push the door it changes position, regardless of the positions of other objects (see JKRULES) and the plan he arrives at is consequently that of pushing the door. When the plan is added to the tree, the tree looks like this:

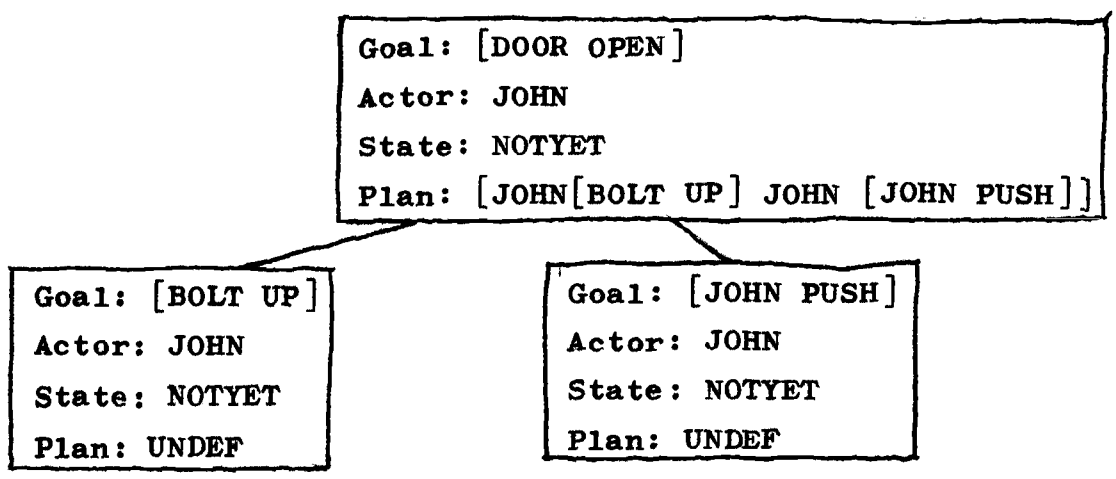


The bottom leftmost goal always becomes the current goal, and this is [JOHN PUSH]. Since it is an event, John carries out the action and marks the state of the goal as ACHIEVED. He then examines the world and finds it unchanged: the door is not open. This contradicts his rule that if you push doors they change position, and he alters the rule. He could either decide that, for the door to change position, the robot must be out, or that the bolt must be up. Suppose he is lucky and that his arbitrary choice is correct. His rule now says

[EVT [ROBOT PUSH] SIT [BOLT UP] RES [DOOR]]

If a goal is achieved or failed it is wiped off the tree; [JOHN PUSH] thus disappears. [DOOR OPEN] is again the current goal, and its state is still NOTYET. This time the plan suggested by John's new rule is

[JOHN [BOLT UP] JOHN [JOHN PUSH]], and since this is not equal to the old, unsuccessful plan of [JOHN [JOHN PUSH]], it is added to the tree. The tree now looks like this:



The current goal is now [BOLT UP] as it is on the bottom left, and since it is not an event another plan is sought. John believes that if you slide the bolt it changes position, so he adds this as his plan:

Goal: [JOHN SLIDE]
Actor: JOHN
State: NOTYET
Plan: UNDEF

This becomes the current goal, and it is an event. Once John has slid the bolt the goal is marked ACHIEVED and the world changes so that the bolt is up. John alters JKWORLD accordingly and leaves JKRULES as it is, since the result of the action was predicted correctly. The state of [BOLT UP] is also marked as ACHIEVED. When all achieved goals have been wiped off, the current goal becomes [JOHN PUSH]. By a similar process this leads to an action, a further change in the world (the door opens) and a corresponding change in JKWORLD. The state of [DOOR OPEN] is marked as ACHIEVED, all goals are wiped off the tree, and the task has been completed.

2.4.3 How plans are made, evaluated and justified

The topics considered in this section were briefly mentioned in 2.3.2; we now go into more detail, and follow through an example stage by stage. The first topic is the process by which a plan is made to achieve a given goal,

and we will use a case from the example in 2.4.2. During that example, there was a point when the tree and the key variables were as follows:

Tree:

Goal: [DOOR OPEN]
Actor: JOHN
State: NOTYET
Plan: [JOHN [JOHN PUSH]]

JKWORLD: [JOHN IN MARY OUT BOLT DOWN DOOR SHUT]

JKRULES: [[EVT [ROBOT MOVE] SIT [ANY] RES [UNDEF]]
[EVT [ROBOT PUSH] SIT [BOLT UP] RES [DOOR]]
[EVT [ROBOT SLIDE] SIT [ANY] RES [BOLT]]]

John produced the plan of [JOHN [BOLT UP] JOHN [JOHN PUSH]] and did so by following these 5 stages:

(i) Find out the goal for which a plan is needed

This is easy, since it is the current goal on the tree, [DOOR OPEN].

(ii) Find out which rule is relevant to the goal

To do this, John searches through JKRULES looking at the RES of each rule and comparing it to the first symbol of the goal, DOOR. Actually, he compares it to the TYPE of the first symbol. (Each object has a "type", as follows: John and Mary are ROBOTS, the door is a DOOR, and the bolt is a BOLT.) So John looks for a RES equal to [DOOR] and finds that the second rule in JKRULES is what he wants. If

he had failed to find a RES equal to [DOOR] he would have proceeded as follows:

(a) if all the RES parts of the rules said [NOTHING] - i.e. if he believed that whatever you do, nothing happens, he would have decided that no plan could be found and marked the goal as [FAILED].

(b) otherwise, he would choose a rule at random and use that as the basis of his plan.

But in our example the rule [EVT [ROBOT PUSH] SIT [ANY] RES [DOOR]] is chosen as the relevant rule for the goal.

(iii) Make the rule specific - i.e. substitute JOHN or MARY for ROBOT.

This stage involves the choice of who is going to carry out the action specified by the EVT of the relevant rule. If John is on his own, there is no choice; but if responsibility for the goal is joint then he can select Mary if he wishes. In the latter case, he first finds out if he can do the action, and if not finds out if Mary can, asking if he doesn't already know. If no-one can do the action, John fails to produce a plan.

In our example, John can push the door and so he substitutes JOHN for ROBOT in the rule and gets the specific rule: [EVT [JOHN PUSH] SIT [BOLT UP] RES [DOOR]]

(iv) Find out if the SIT mentioned in the rule is achieved

The SIT in our example is [BOLT UP], and it is not achieved.

If the SIT had been [ANY] or [BOLT DOWN] it would have been achieved. A SIT of [ANY] is always achieved by definition since it means "whatever the circumstances", but a SIT like [BOLT UP] has to be tested by looking at JKWORLD. If John and Mary were working jointly and John didn't know whether the bolt was up, he would ask a question at this point to find out. As it is, he can find out that the SIT is not yet achieved by consulting his memory.

(v) Use the information gathered in (ii)-(iv) to make a plan

The plan will always involve an event, and may or may not have a situation preceding it. The event is the EVT in the specific rule found in (iii): [JOHN PUSH]. Events are always the responsibility of the robot specified in the first symbol. So the plan is so far [JOHN [JOHN PUSH]].

If the SIT is achieved (see (iv)) there is no need to add a situation; if not (as in the example), the SIT is made into a subgoal to precede the event. If the goal was joint, the actor responsible for the situation will be BOTH - the subgoal will be joint too; but if (as here) only one robot is responsible for the goal, he is also responsible for the situation. So the final plan is [JOHN [BOLT UP] JOHN [JOHN PUSH]]; in English, John is going to get the bolt up and then push the door.

The evaluation and justification of plans are relatively simple matters and can be dealt with quickly. All plans contain events, and the first step in the above two processes

is to find out which rule in JKRULES has the equivalent event (or EVT). Suppose the plan is [BOTH [BOLT UP] JOHN [JOHN PUSH]] and John is evaluating. He first checks that he can do the action assigned to him; if not, he rejects the plan. Then he finds the relevant rule, the rule with an EVT containing the action PUSH. He constructs an imaginary world by using JKWORLD, updated by bringing it into line with the situation [BOLT UP], and then uses the rule to see whether, in the world after [BOLT UP] is achieved, the event [JOHN PUSH] will alter the position of the door, the object mentioned in the main goal [DOOR OPEN]. If so, he accepts the plan; if not, he rejects it.

If asked to justify a plan, John must find out which rule he based it on. He does this as explained above. If the rule has a RES which is irrelevant to the goal, he assumes that it was chosen randomly (see (ii) in the description of how plans are made) and makes no effort to justify the plan. But if the RES is relevant, he justifies the plan by stating the whole rule. For example, he says "If you push the door when the bolt is up, the door opens" which is the English translation of

[EVT [ROBOT PUSH] SIT [BOLT UP] RES [DOOR]]

(Note that the "you" referred to is not Mary, but really means "one".)

2.5 Routines

2.5.1 General nature of routines

In 2.4.2 and 2.4.3 we gave fairly detailed examples of the processes by which plans are used and made, but said nothing about how these processes are carried out. The answer is that they are carried out by ROUTINES, which we now consider. To begin with, it is necessary to discuss what problems arise in constructing a system which has to break off its thoughts now and then so as to make an utterance, and then both comprehend the reply and carry on with its thinking. What is special about routines is that they can be broken off and then re-entered later at the correct place; and that they can be conveniently linked to procedures which carry out a conversation.

Programs consist of two kinds of structure: data structures and procedures. Procedures are lists of instructions which alter the data structures and sometimes communicate with special devices (such as teletypes). The distinction is not a sharp one, as procedures are also lists of symbols, and there is no reason why they should not be altered by other procedures as if they were data structures. In POP-2, the main kind of procedure is the FUNCTION. When a function is called, a space is made for a number of local variables declared in the function definition, and the instructions in the function body are executed in sequence.

It is important to note that the local variables are tied to a particular call of the function, and if the function is called again (perhaps recursively) the second set of variables is distinct from the first. As soon as a function is over, the local variables associated with that call are lost.

Functions can call other functions: in other words, an instruction in function A can invoke a call of function B. As soon as B is finished, control returns to the next instruction in A. If it is desired that B should send a value back to A the value is left on a structure called the stack, which is a place for piling up messages. So B can put its message on the stack, and a later instruction in A can collect it.

Routines share most of these features, often realised in a different way; in addition, they are stored in lists, which are available to the program as data structures, and when they are called, the control structure is also a list and also available as data. By "control structure", we mean the list which contains the names of the routines called, their local variable values, and the instruction reached in each. So it is possible for an instruction to have direct access to the structure which called it, and it is also possible to exit from the program as a whole without losing place of where you are in the routines: this information survives as data.

The shortest routine is the one which makes plans, JRPLAN. Its definition is written like this:

```
ROUTINE JRPLAN;
1. * JRGOAL;
2. * JRRULE;
3. * JRSPEC;
4. * JRSIT;
5. ↑ JRCOMP;
END;
```

Some of this definition is punctuation, to make it easier to read and interpret. The system makes a list from the definition and assigns the list to the variable JRPLAN - the name of the routine. This list is then the definition of the routine as it will be consulted by the rest of the program. In this case JRPLAN will be given the value: [[1 *JRGOAL][2 * JRRULE][3 * JRSPEC][4 * JRSIT][5 ↑ JRCOMP]]. There are five instructions here, each of three parts. The first part is the number of the instruction and will be used to mark the place that has been reached in the routine. The second and third parts need rather more explaining.

It was claimed above that routines duplicated most of the features of functions. The definition given above, however, contains no local variables. This is because local variables are realised in a different way: they are

associated with the instructions whose job it is to determine their value, it being understood that this value will be consulted by subsequent instructions. It is a feature of routines that local variables, once given values, retain those values until the routine exits. The sign * indicates that the instruction concerned is designed to produce a value, and since instructions 1-4 all fall into this category, when JRPLAN is called four spaces (numbered 1-4) will be set up in which to store these values once they are found. The sign ↑ indicates that no variable is associated with the instruction concerned (5 in this case). This usually implies that the instruction will be using the values found in 1-4 to produce some side-effect. We shall use the term ENTRIES to refer to the values associated with instructions marked * .

The last symbol is the name of a function: JRGOAL, JRRULE, etc. are all defined elsewhere as functions. Commonly a function makes the entry for its instruction. For instance, instruction 1 specifies the function JRGOAL which finds out the goal for which a plan is required. This goal (e.g. [DOOR OPEN]) will be the entry for instruction 1, will be stored as such, and available to JRRULE, JRSPEC, etc. if need be.

Simple Example of how Routines work

To show how routines can be used to carry out computations,

we consider a trivial arithmetic example which is not in the actual system, but which shows how routines can call other routines, and how they can be interrupted and still come back in at the right place. The computation is this: the program will accept a number from the operator, square it, and divide by two. The routine which does all this will call another routine which only accepts a number and squares it. First we give the routine definitions:

```
ROUTINE HALFSQ;          ROUTINE SQUINPUT;
1. * SQ;                 1. * FINDX;
2. ↑ HALVE;              2. ↑ SQX;
END;                     END;
```

Two variables will be created, HALFSQ and SQINPUT, and they will hold the definitions

```
[[1 * SQ] [2 ↑ HALVE]]     (HALFSQ)
```

and `[[1 * FINDX][2 ↑ SQX]]` (SQINPUT)

To call routine HALFSQ, spaces to store the instruction reached and the entry for 1 must be made, and the variable which holds these values we will call CONTROL, since it controls the computation. When HALFSQ is called, the following association list is put into CONTROL:

```
[[NAME HALFSQ PLACE 1 ENTRIES [1 NIL]]]
```

The first two pairs tell us that the routine running is called HALFSQ and that the place reached in this routine is 1. The list after ENTRIES contains a space to put the entry

for instruction 1. That space is now empty, this being signified by NIL.

We now refer to a few basic functions needed to manipulate CONTROL. We need a function to make an entry, one to find out which entry is in a given place, and functions to call a routine or to exit from one. These will be called ENTER, ENTRY, CALL and EXIT.

ENTER takes one argument, the entry, and puts it in the ENTRIES list after the current place. So if the PLACE is 1 (i.e. we are on the first instruction), ENTER (26) puts 26 in place of NIL in the list CONTROL mentioned above.

ENTRY takes as argument an instruction number and delivers the associated value in ENTRIES. So ENTRY (1) will be NIL given that CONTROL is the value mentioned above, but 26 after ENTER(26) is called.

CALL takes as argument the routine to be called. If CONTROL was empty ([ ]), CALL ("HALFSQ") would give it the above value.

EXIT takes no argument: it removes the top list from CONTROL, thus reversing CALL.

We can now give actual POP-2 definitions of the functions mentioned in the definitions of routines HALFSQ and SQINPUT.

```
FUNCTION SQ;
CALL ("SQUINPUT");
END;
```

```
FUNCTION HALVE;
SAY (ENTRY (1)/2);
END;

FUNCTION FINDX;
IF INPUT = UNDEF THEN SAY (WHAT IS YOUR NUMBER);
ELSE ENTER (INPUT) CLOSE;
END;

FUNCTION SQX;
VARS X; ENTRY (1) --> X;
EXIT ( ); ENTER (X*X);
END;
```

The function SAY here exits from the whole program (hot the routine) and outputs its argument. The variable INPUT is used to hold the number the operator wants squared, and is initially set to UNDEF. Finally, we will assume that a function RUN can be used by the operator to set the whole program going.

At the start, HALFSQ and SQINPUT will have the values given below, INPUT will be UNDEF and CONTROL will be [ ], an empty list. The operator first types CALL("HALFSQ") which loads the routine. CONTROL will now be [[NAME HALFSQ PLACE 1 ENTRIES [1 NIL]]]. Then he types RUN() to set the program in motion. First it finds out if the entry for 1 is filled: it is not. So the interpreter (henceforth E) finds which instruction occurs in place 1 of HALFSQ: it

turns out to be [1 *SQ]. The function SQ is now called. The result is that the command CALL ("SQINPUT") is carried out (see definition of SQ) and the routine SQINPUT is loaded into CONTROL on top of HALFSQ:

CONTROL: [[NAME SQINPUT PLACE 1 ENTRIES [1 NIL]][NAME HALFSQ etc..

E goes back to CONTROL and, by an exactly similar process, calls function FINDX, from the first instruction of SQINPUT. FINDX finds that INPUT does equal UNDEF and thus executes the command SAY("WHAT IS YOUR NUMBER"). SAY prints its argument: "What is your number", and RUN exits, control returning to the operator. The routine SQINPUT has arranged for something to be said, and then allowed the program to exit, the idea being that the operator will put his number into INPUT and call RUN again. The routine will then carry on from where it left off, having the information it needs.

So the operator types, say, 12 → INPUT; RUN();
and by the same process, FINDX gets called again. This time INPUT is not equal to UNDEF and 12 is entered in the current place, i.e. 1.

E now finds that the entry for instruction 1 is made, and puts PLACE up to 2. CONTROL is now;

[[NAME SQINPUT PLACE 2 ENTRIES [1 12]][NAME HALFSQ ...]]

E now looks at instruction 2 of SQINPUT and calls the function SQX. This first gets the entry made by SQX and

assigns it to a local variable X, which is thus 12. Then it calls EXIT, which takes the top list out of CONTROL. Thus SQINPUT has been removed from CONTROL by its own subsidiary function before that function has finished. Since SQINPUT is gone, the call of ENTER puts the square of X (i.e. 144) into the entry for the current place of HALFSQ, this now being the top routine. It is in this manner that a routine can send a value back to the routine which called it. CONTROL is now [[NAME HALFSQ PLACE 1 ENTRIES [1 144]].

E sees that the entry for 1 is now filled, and alters PLACE to 2. Function HALVE is then called and it divides the entry of instruction 1 (144) by 2, outputs the answer (72) and jumps out of RUN. Control returns to the operator and the computation is done.

It is hoped that this example will convey a feel for how routines work, especially with regard to the possibilities of interrupting the computation to say something, and the way in which subsidiary functions can have access to the structure which controls them.

2.5.2 Routine BASIC

John has three routines: JRBASIC, JRACHIEVE and JRPLAN. JRBASIC arranges for the main goal to be attempted, with or without Mary's help, and decides what to do once the attempt has been made. JRACHIEVE is called by JRBASIC, and its job is to achieve the main goal: it is told by JRBASIC whether the attempt is sole or joint, and reports its success

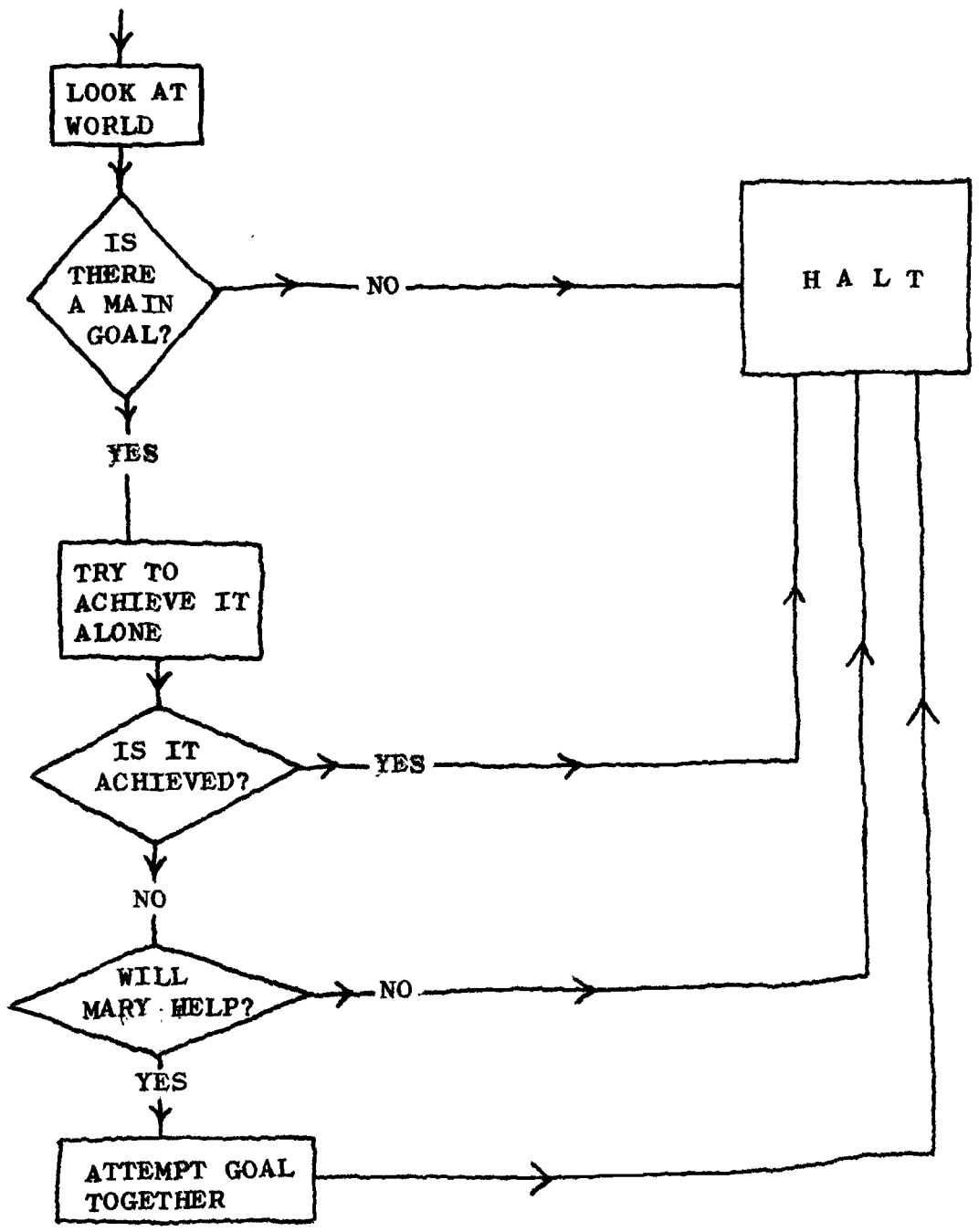
or failure back to the instruction in JRBASIC which invoked it. JRPLAN is called by JRACHIEVE to find a plan to achieve a particular goal, either the main goal or one of its subgoals. It returns a plan or a failure message.

The easiest way to explain JRBASIC is to say exactly what it does, since there is no convenient phrase which captures its purpose. It is loaded before every run of the program, and never exits; hence the name "BASIC". When the main goal is achieved or abandoned it goes into a "halt" state from which it returns control to the chairman every time John is "aroused". We will explain its behaviour twice: first in general terms, and then by giving its definition and taking it instruction by instruction.

The first thing JRBASIC does is to look at the world and see where the objects are; in other words, to initialize JKWORLD. Then it finds out whether there is a main goal, and if not, goes into the halt state (halts). If there is a main goal, JRBASIC sets up JRACHIEVE to attempt the goal without Mary's help (John tries to bring about the goal on his own). If the attempt succeeds, JRBASIC halts; if not, it arranges for an appeal to be made to Mary. (John asks Mary whether she will help him achieve the goal.) Mary will either say "Yes" or "No"; if she says no, JRBASIC halts; but if she says yes, JRACHIEVE is called again, this time with the main goal being a joint

responsibility. When JRACHIEVE has achieved or failed, JRBASIC halts, there being nothing further to do in either case.

It might help if the above account is put into flow-chart form:



The definition of JRBASIC is:

```
ROUTINE JRBASIC;
1. ↑ JRPREP;
2. * JRMAINGL;
3. ↑ JRPREP1;
4. * JRACHGL;
5. ↑ JRMOVE1;
6. * JRAPPEAL;
7. ↑ JRPREP2;
8. * JRACHGL;
9. ↑ HRHALT;
END;
```

On its own, this definition is not too enlightening, but when the instructions are taken one by one it should become clear that the definition corresponds closely to the flow chart given above.

1. ↑ JRPREP

Prepares the memory by looking at the world and updating JKWORLD accordingly, then initialising the other variables.

2. * JRMAINGL

Finds the main goal and enters it. (The main goal will have been put into JKGOAL by the operator.) If the goal is, say, to get John in, the entry will be [JOHN IN]; if there is no goal, it will be [NONE].

3. ↑ JRPREP1

If entry 2 is [NONE] (there is no goal) it goes to 9 (i.e. halts). If entry 2 is a goal, the tree is set up for a sole attempt on the goal by John. Setting up the tree means filling in its top node, e.g.

Goal: [JOHN IN]
Actor: JOHN
State: NOTYET
Plan: UNDEF

4. * JRACHGL

Calls routine JRACHIEVE which tries to achieve the goal without help. It knows that only John is involved because the actor of the main goal was set (in 3) to JOHN, and not to BOTH. JRACHIEVE returns [ACHIEVED] if it succeeds, and [FAILED] if not, these being the possible entries for JRBASIC 4.

5. ↑JRMOVE1

If entry 4 is [ACHIEVED], it goes to 9 (halts); if not, it goes to 6. To "go to" an instruction all it has to do is to make the instruction the next to be attempted by the control. It will be recalled that control information is accessible to the program, so it just has to go through the control list and change the appropriate number.

6. * JRAPPEAL

Calls a procedure to ask Mary if she will help. Procedures

which carry out conversations are called GAMES and are described in the next section (6). The game will enter Mary's answer - [YES] or [NO] - in JRBASIC 6. (i.e. in the instruction which called it).

7. ↑ JRPREP2

If Mary said [NO] (i.e. entry 6 was [NO]) it goes to 9 (halts). If [YES], it prepares the planning tree for a joint effort at the goal. The planning tree will look like this:

Goal: [JOHN IN]
Actor: BOTH
State: NOTYET
Plan: UNDEF

8. * JRACHGL

Same as 4, except that JRACHIEVE will find that the goal is to be attempted jointly (the actor is BOTH). The entry is again [ACHIEVED] or [FAILED], though in this case it is not used: whichever message is entered, the program goes into JRBASIC 9 and halts.

9. ↑ JRHALT

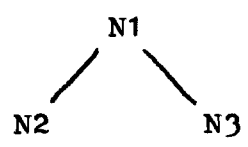
Returns control to the chairman. This is called "swapping" as it usually involves giving control to Mary. If Mary has reached MRHALT she will swap control back, and if this happens (i.e. if there are two consecutive swaps) the chairman function exits and the run is over.

2.5.3 Routine ACHIEVE

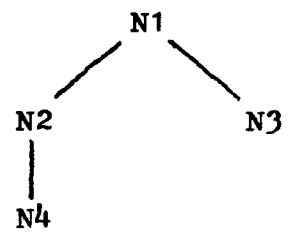
The routine JRACHIEVE is called by JRBASIC 4 or 8, and returns [ACHIEVED] or [FAILED]. It is given a planning tree with the main goal specified, and its job is to develop the tree, perform such actions as the tree mentions, and return only when the state of the main goal is ACHIEVED or FAILED, rather than NOTYET. It would seem that such a procedure would have to be recursive, and indeed there is a sense in which JRACHIEVE is just that. But the recursive effect is attained by a method which avoids the necessity of piling up calls of JRACHIEVE to the depth reached in the tree. When one version of JRACHIEVE calls another, it destroys itself first, so that there is never more than one layer. This is possible because all the information that the second call requires is kept independently on the tree. A single run of JRACHIEVE takes the tree it is given and develops it, then exits and calls a new version of JRACHIEVE which in turn deals with the tree it has inherited. If one of the versions of JRACHIEVE finds that the main goal is achieved or failed, it returns this result to JRBASIC, and, for the present, no more calls of JRACHIEVE are made.

We will illustrate this by an example in which nodes on the tree will be designated by N1, N2, N3, etc. and calls of JRACHIEVE by A1, A2, A3, etc. Suppose that JRBASIC puts the main goal N1 on the tree and then calls A1,

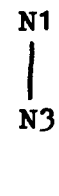
the first JRACHIEVE. A1 starts with a tree containing only N1, and perhaps it finds a plan involving the subgoals N2 and N3, and puts these on the tree:



It then exits and calls A2. A2 inherits the larger tree, and develops the node N2, handing over this tree to A3:



Suppose N4 is an action. A3 will carry out the action, achieve N4, and perhaps achieve N2 as well. It wipes these achieved nodes off the tree and hands over this tree to A4:



A4 performs action N3, achieves both N3 and N1, and since N1 is the main goal returns [ACHIEVED] to JRBASIC.

As this example shows, there are two main things that a call of JRACHIEVE can do to the tree:

(i) it can develop a node;

(ii) it can perform an action and prune the tree accordingly.

Whichever it does, it must hand over a tree which consists entirely of goals marked NOTYET. No plan with achieved or impossible goals will ever be added, and after an action, all achieved or failed goals must be wiped off the tree in readiness for the next call.

We now give the definition of this routine, followed by a step by step description, and a rough flow chart.

```
ROUTINE JRACHIEVE;
1. * JRCURRGL;
2. * JRACTOR;
3. * JRKIND;
4. * JRSTATE;
5. * JRPLAN1;
6. ↑ JRRETURN;
7. * JRBEFORE;
8. * JRACT;
9. * JRRESULT;
10.* JRPARENT;
11.* JRLESSON;
12.↑ JRPRUNE;
END;
```

1. * JRCURRGL;

Enters the "current goal", i.e. the next node to be

attended to. This is always the bottom left node, provided that the tree is properly pruned (i.e. all goals are marked NOTYET). Appropriate entries would be [DOOR OPEN] or [JOHN PUSH] (etc).

2. *JRACTOR

The entry is the actor responsible for the current goal, e.g. [JOHN] or [BOTH].

3. * JRKIND

Enters whether the current goal is an event ([EVENT]) or a situation ([SITN]). [JOHN PUSH] would be an event, [DOOR OPEN] a situation. If it is an event, we go to 7, since instructions 7 - 12 are concerned with carrying out actions and assessing their consequences. If it is a situation, we go on to 4, since 4 - 6 are concerned with adding a plan to the tree (i.e. developing the node).

4. * JRSTATE

Enters the state of the goal, asking if necessary. The state will be marked as NOTYET on the tree, but this instruction makes sure by examining the world model, and alters the tree if it gets a different answer. If, say, the goal is [DOOR OPEN] and John doesn't know the position of the door, he sets up a game to ask Mary the appropriate question ("Is the door open"), so that when JRSTATE is recalled the information will be available. Possible entries are [ACHIEVED], [FAILED] and [NOTYET], usually of course the

last of these.

5. *JRPLAN1

The entry is either a plan, or [NO] if none was found or agreed on. First, entry 4 is checked, and if it is not [NOTYET] we go to 12 to prune the tree. Otherwise, John either has to choose a plan himself (if the goal is his sole responsibility), or arrange for a plan to be agreed (if the goal is joint). To choose the plan himself, he calls the routine JRPLAN; if a plan has to be agreed, he arranges for a game to be set up through which the plan will be chosen jointly. In either case, the routine or game will return an appropriate entry - e.g. [NO] or [JOHN [JOHN SLIDE]] or [BOTH [DOOR OPEN] MARY [MARY MOVE]].

6. ↑ JRRETURN

If no plan was found (i.e. entry 5 was [NO]), the current goal is failed and we go to 12 to prune the tree. Otherwise the plan in 5 is added to the tree, the present version of JRACHIEVE exits, and a new one is reloaded to handle the new tree with its new current goal.

7. * JRBEFORE

A preparation for the action which is about to occur. Records the positions of the objects (JKWORLD) so that once the action has occurred, and JKWORLD has been updated, it will be possible to compare the world before and after the

action and assess its consequences. A possible entry is [JOHN IN MARY OUT DOOR SHUT BOLT UNDEF].

8. * JRACT

If it is John's action, he performs it, enters [DONE], and tells Mary he has done it. If it is Mary's action, John swaps control to her; she does it, tells him, and he enters [DONE] as a result of being told. In either event, the goal is marked as ACHIEVED on the tree.

9. * JRRESULT

If John and Mary are co-operating, the result of the action is assessed jointly by a game. If John is on his own, he looks again at the world, and compares it with the situation recorded in entry 7. The entry is [NOTHING] if no change is observed, or the name of the object which changed position, e.g. [JOHN] or [BOLT].

10. * JRPARENT

Similar to 4. Finds the state of the parent of the current goal, and puts it on the tree. As in 4, if John finds that he doesn't know the relevant object position, he asks, and the state is eventually made the entry: [ACHIEVED], [FAILED] or [NOTYET].

11. * JRLESSON

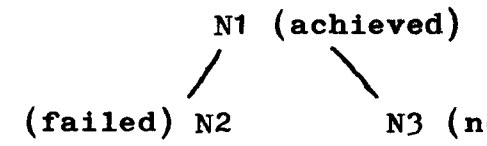
Uses the knowledge of the action performed, and its consequences, to update the theory of how the world works (JKRULES). If John's rules predicted a different outcome

from that recorded in entry 9, he makes a new rule by the process described at the end of section 2.3.2. If the formation of a new rule requires information from Mary, or if John finds that his rule worked while one he attributes to Mary failed, then a game is arranged to conduct the appropriate conversation. (John will aks a question in the first case, and explain a rule in the second.) The entry, once all these tasks are done, is [LEARNED], signifying that John has learned the lesson from his experience.

12. ↑ JRPRUNE

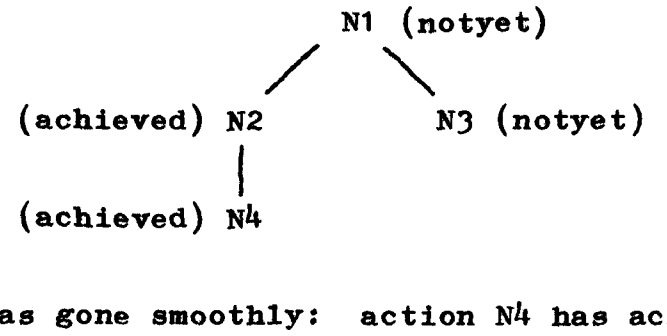
This instruction first examines the state of the main goal, and returns to JRBASIC if the state is ACHIEVED or FAILED. If the state is NOTYET, the tree is pruned for another run of JRACHIEVE. All achieved or failed goals are wiped off the tree, and if a subgoal fails, any subgoals which are part of the same plan are wiped off with it. We give four examples of trees before and after being treated by JRPRUNE:

(i)



Since the top goal is achieved, JRPRUNE kills JRACHIEVE and [ACHIEVED] is returned to JRBASIC.

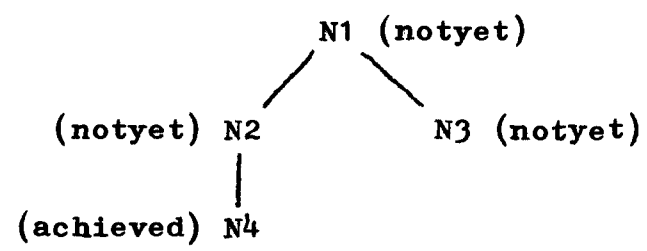
(ii)



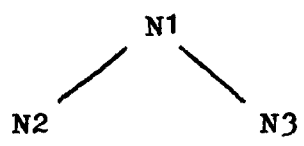
Here all has gone smoothly: action N4 has achieved goal N2.

The tree is pruned to

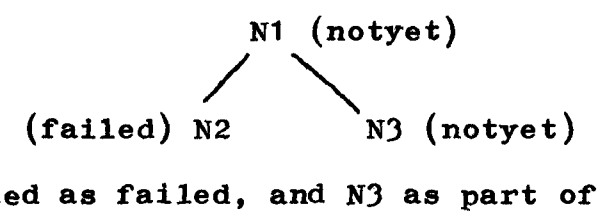
(iii)



N4 has failed to achieve N2, and the tree is pruned to:



(iv)

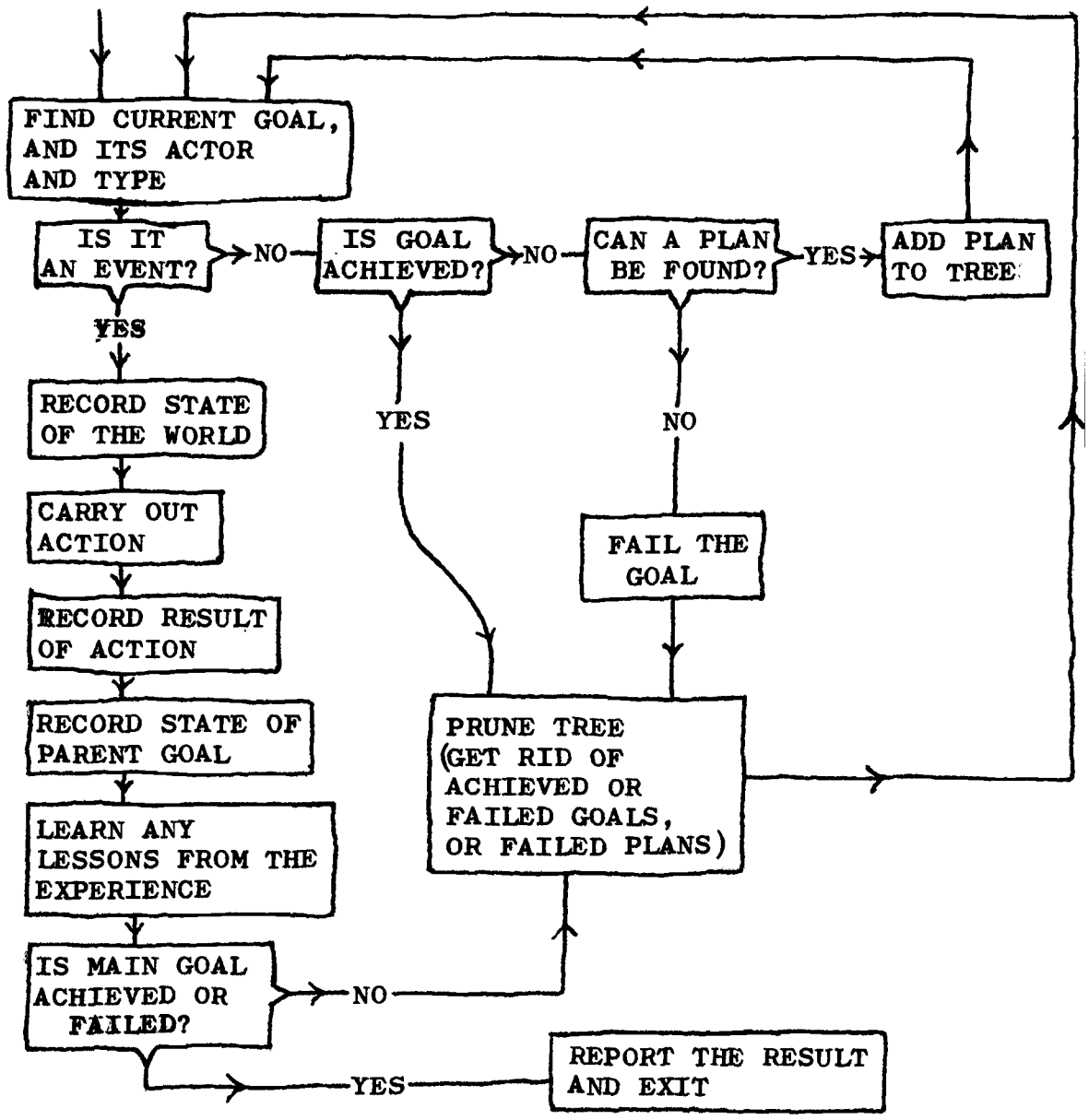


N2 is discarded as failed, and N3 as part of a now hopeless plan; only N1 is left

N1

Having pruned the tree, JRPRUNE kills JRACHIEVE and reloads it for a fresh run, beginning at the first instruction and with empty entries.

The flow chart for this process might be represented thus:



2.5.4 Routine PLAN

The routine JRPLAN tries to make a plan, and returns (to the instruction which called it) [NO] if it fails, and a plan if it succeeds. The general procedure has already been described in sections 2.3.2 and 2.4.3, so we give only the definition, the step by step description, and a flow chart.

```
ROUTINE PLAN;
1. * JRGOAL;
2. * JRRULE;
3. * JRSPEC;
4. * JRSIT;
5. ↑ JRCOMP;
END;
```

The instructions 1 - 5 correspond exactly to steps (i) to (v) in the description given in 2.4.3, so a comparison may be useful.

1. * JRGOAL

The entry is the current goal, the one for which a plan is required (e.g. [DOOR OPEN] or [JOHN IN]).

2. * JRRULE

The entry here us the rule which is relevant to the goal: the one which shows how the position of the object in question can be altered. If none of the rules help, the choice is

made arbitrarily, unless it is known that all actions yield no result, in which case JRPLAN exits and returns [NO]. A possible entry (for the goal [DOOR OPEN], say) is [EVT [ROBOT PUSH] SIT [BOTL UP] RES [DOOR]].

3. * JRSPEC

Finds out who is to carry out the action (i.e. push the door in the above example) and makes the rule specific to that person. If John can't do it, he makes sure Mary can (asking if necessary) before going ahead. If neither of them can do it, or John is on his own and cannot do it, JRPLAN exits and returns [NO]. If, in the example used in 2, Mary can push the door but John can't, the entry will be:

[EVT [MARY PUSH] SIT [BOLT UP] RES [DOOR]]

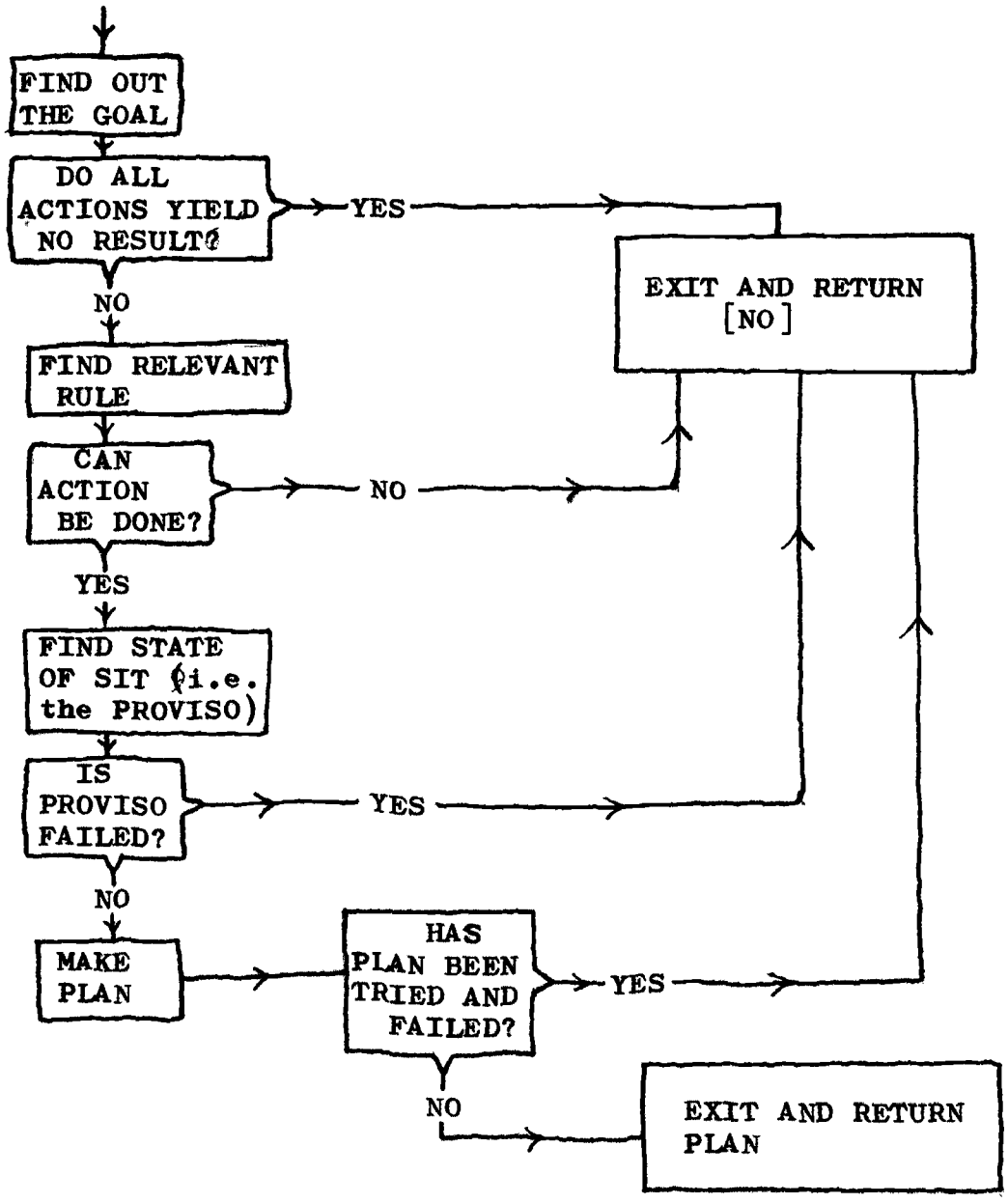
4. * JRSIT

Enters the state of the SIT in the rule in 3, asking if necessary. The entry will be [ACHIEVED], [FAILED] or [NOTYET].

5. ↑ JRCOMP

Completes the plan out of the information gathered in 3 and 4. If the entry for 4 is [FAILED], or the plan arrived at has already been tried and not worked, JRPLAN exits and returns [NO]. Otherwise,the event after EVT in 3, and the SIT as well if it is not achieved, are combined to make a plan. In the case mentioned above (in 3) the

plan would be [BOTH [BOLT UP] MARY [MARY PUSH]] if [BOLT UP] was NOTYET (see 4), and [MARY [MARY PUSH]] if it was ACHIEVED. Finally, JRPLAN exits and returns the completed plan. The flow chart, roughly speaking, is:



2.6 Games

2.6.1 General

It will be recalled that the basic problems we are concerned with are (a) why do we conduct conversations, and (b) how do we conduct them. A possible answer to the first of these problems can be assembled from the previous sections. We have described a system which has "global" models of its world, and the rules by which it works; we have shown that it can use these models as it constructs and carries out plans; and we have shown that in the process of making plans and putting them into operation, occasions arise where a decision has to be made jointly, or one robot requires information from the other, and so on. Lastly, we have described a kind of procedure, the "routine", which enables the program to carry out the planning processes and to keep track of the place it has reached in these processes, so that conversational diversions can occur without the underlying thought processes being lost or disrupted. Routines, then, constitute the mental processes which underlie the robots' conversation, and it is the routines which determine when a conversation will occur, what kind of conversation it will be, and how its outcome is to be made use of. If we want to know why John initiates a particular conversation, we will find the answer by examining his routines: which one he is in, and which

point he has reached.

Games, on the other hand, are the structures which conduct the conversations. They are procedures, and are called by routines, with which they have a great deal in common. The major difference is that a game, as its name suggests, is a procedure which is conducted by two parties (or players),not by one. If Mary was not around, John could not conduct a game. Games specify the form that the conversation should take: what should be said, and who should say it. There are two roles in each game, called "White" and "Black" by analogy with chess and draughts, and each utterance is thought of as a move in the game. There are definite rules as to what counts as a proper move at the various stages of each game, and if one of the players breaks a rule (says something inappropriate) his partner will interrupt the game and declare that a muddle has arisen. There are seven games, one for each conversational purpose that the robots meet with:

(i) JGASK: Game to get information

(ii) JGTELL: Game to give information

(iii) JGRULE: Game to explain and discuss rules (theories)

(iv) JGGOAL: Game to ask for help with a goal

(v) JGPLAN: Game to agree on a plan

(vi) JGASSESS: Game to assess the result of an action

(vii) JGGAME: Game to arrange one of the games (i) to (vi)

Each game consists of several utterances; JGASK, for example, has two: 1) White asks a question 2) Black answers it. For a game to be properly played, each player must know which game it is, which colour he is, and which move has been reached. Only then can the robots make appropriate utterances, and give the right interpretations to the utterances of each other.

As an example, we will discuss one of the simpler games, JGASK, and show what it looks like and how it works. First, its definition:

```
GAME  JGASK;
1. * W JGQUERY [JDQUERY];
2. * B JGANSWER [JDSIGN];
3. ↑ W JGRECORD;
END;
```

This definition illustrates various features of games:

(a) The structure is very similar to that of routines, and indeed the representation inside the computer is the same. A variable JGASK is set up, and given the value

```
[[1 * WHITE JGQUERY [JDQUERY]][2 * BLACK JGANSWER [JDSIGN]]
    [3 ↑ WHITE JGRECORD]]
```

Just as routines can be loaded on top of each other, in control, (i.e. can call each other), so games can be called by routines, and can also call either routines or other games.

In other words, the control structure of a robot can consist of any mixture of games and routines, provided that JRBASIC is at the bottom of the pile. (By this we mean that the system can handle any mixture, not that any mixture can arise during a run.)

(b) Instructions are numbered, as with routines, so that the robots can keep their place in the game.

(c) The symbols * and ↑ again indicate that the instruction is or is not tied to a local variable, i.e. to an entry. But in games, the * symbol has an extra significance in that all entries are uttered by the robot making them. If John asks a question, his question is the entry for instruction 1; but since it is a joint procedure, Mary must also make the entry, so John must say what it is.

(d) The symbols W and B stand for WHITE and BLACK and indicate who is to carry out the instruction. If John is White, he will take the first instruction and make the entry (his question); he will also utter it. Mary then takes the second instruction, since she is Black, and utters her answer, the entry for 2. Finally, John carries out instruction 3, in which he updates his memory in line with Mary's answer; since it is a ↑ instruction there is no entry and hence no utterance.

(e) As with routines, each instruction contains a function which carries out the associated tasks. JGQUERY arranges

for the question to be asked; JGANSWER finds the answer and arranges for it to be given; JGRECORD uses the first two entries to update the memory.

(f) All instructions marked * also have lists of symbols at the end, e.g. [JDQUERY] for 1 and [JDSIGN] for 2. These have a double significance:

(i) they represent the rules for what utterances are appropriate.

(ii) they determine how the utterance is processed.

To take the case of instruction 2: having asked a question in 1, John must be ready to make sense of the answer. JDSIGN is a function which accepts a list of English words and tries to analyse it as a "sign" (i.e. "Yes", "No" or "I don't know", roughly speaking). If it succeeds, it returns a translation of the utterance into a standard internal language; if it fails, it returns UNDEF. Should every function in the list return UNDEF, Mary's utterance will be judged inappropriate and John will interrupt the game and tell her so.

The entry, then, is not an English expression but an expression in a more concise internal language, and there are functions which translate from one language to the other in either direction. To translate from the internal language to English, there are functions corresponding to

every utterance in every game. Thus for JGASK, there are two such functions, JWQUERY and JWANSWER. To translate the other way, there is one function for every kind of utterance (e.g. query, sign, plan, fact, event, situation, rule, game) which when given a piece of English tries to make it into a query, sign, or whatever, and returns the translated version if it succeeds and UNDEF if not.

In the case of instruction 2, for example, there are three possible entries, [YES], [NO] and [UNDEF]. When Mary gives her answer (say [UNDEF]) she calls MWANSWER with [UNDEF] as argument, and gets an English translation which becomes her utterance: [I DONT KNOW]. To try and make sense of this, John will try every function in his list for instruction 2; in this case there is only one, JDSIGN. JDSIGN takes [I DONT KNOW] as argument and returns [UNDEF], which becomes the entry. There are only four things JDSIGN can return: [YES], [NO], [UNDEF], UNDEF. The first three mean "The utterance was a sign and this is the sign it was", and the last means "The utterance cannot be construed as a sign so I don't know what it is."

So far, we have discussed the game on the assumption that it is already loaded into the control structures of both robots, and it is now time to say how the loading takes place. When John wants to play JGASK, how does Mary know that this game is to be played? In real life there are two

ways of starting a conversation; the first is to say explicitly that you want to (e.g. "May I ask you something") and the other is to go straight ahead with the first move ("What is the time?"), leaving the other person to infer that you want to ask something. If A meets B and asks "What time is it", and B replies "Go away, I'm busy", he is not replying to the stated question, but to the implied suggestion that they indulge in the question-and-answer "game" at all.

We have chosen to use the explicit method, for two reasons: (i) it is easier to arrange, and (ii) it makes the structure of the conversation clearer, since there are no implied utterances or answers to implied utterances. So John never asks a question outright; he always announces his intention beforehand, thus telling Mary that she is to load MGASK and to take the colour Black. She will then be ready to receive John's question, the first White move in JGASK.

The game JGGAME is the means by which John can suggest that some other game be played. The first move in JGGAME is the name of the game to be played, e.g. ⌈[JGASK]. The reply is a sign indicating that Black is ready to play. Clearly there has to be some special way of loading JGGAME, and it is this:

(i) If the robots are co-operating, and no game is in

operation, any utterance is automatically assumed to be the first move in JGGAME, and on noticing it a robot loads this game and takes Black.

(ii) If a game is already in operation, or the robots have not yet agreed to co-operate, JGGAME has to be loaded by calling the other's name. A call and its acknowledgement are the only utterances which are not moves in any game. If John wants to play the game JGASK when (say) JGPLAN is already going, he must say "Mary!", whereupon she will load MGGAME, take Black, and reply "Yes" (which is also not a move in a game and is ignored). John then goes ahead with the first move in JGGAME, the suggestion that they play JGASK; Mary says when she is ready to; and finally JGASK is loaded by both and the question actually gets asked.

In short: the basic rule is that to get a game loaded, you play JGGAME, and to get JGGAME loaded you call the name of your partner. The only exception arises when (a) you are co-operating to achieve a goal, and (b) no game is in progress; in this case JGGAME can be assumed loaded. The point of this exception is to cut down the amount of calling that goes on: without it, the conversation becomes so long-winded and repetitive as to be tiresome. There is also, perhaps, a certain plausibility in using calling for

interruptions and to make contact, but not for cases where the parties are already co-operating and no other conversation is in progress.

There is one other special feature of JGGAME that has to be mentioned. When White suggests a game such as JGPLAN, i.e. when he suggests that they agree on a plan, it may be that Black will not be ready to do so, having not reached the point in her routine where one looks for a plan. To handle such cases, there is a facility which enables her to hold up the game, continue with the routine until the appropriate point has been reached, and then revert to the game, and agree to play JGPLAN. As a result, when JGGAME returns a plan it will be entered into the right place in Black's routine, as well as White's. The holding up process will be described in detail later, but the basic idea is that the game is saved in another variable, and a mark is put in the routine at the point one wishes to reach. Then control reverts to the routine until the mark is reached, whereupon the game is put back on top of control and the process continues normally.

In the next few sections, we describe the games one by one, with reference to an example which involves all of them. The example is a particular run which is fully described in 2.6.2. First, in 2.6.2, we illustrate the general points

about games made in this section by going through the first few utterances in great detail. Then, in 2.6.3 to 2.6.9, the games are described individually, and illustrated by snatches of dialogue occurring later on in the example run.

2.6.2 An Example

In this section, and the ones which follow, we will use a particular run as an example; we call it R. What distinguishes R from other runs is that its starting position is different, and as a result the conversation turns out differently. The starting position is defined by the initial values given to certain variables, and we give these later. There are two pieces of computer output corresponding to R, which we call R1 and R2. R1 is a record of the dialogue, together with any changes in the world that arise due to actions. R2 also records the dialogue and the changes in the world, but in addition it traces the routines and games, saying when they are loaded, when they end, when each instruction is called, when each entry is made, and what the entries are. We shall refer constantly to R1 and R2 during the examples, and they can be found in Appendix I.

A run is carried out by first initialising certain key variables, and then calling the chairman function which arouses John and Mary alternately until neither has anything further to say. The initial values characteristic of R are as follows:

(a) WORLD

WOBJECTS - [JOHN OUT MARY IN BOLT UP DOOR SHUT]

(b) JOHN

JKGOAL - [JOHN IN]

JKSEE - [JOHN DOOR BOLT]

JKACTS - [MOVE SLIDE]

JKRULES - [[EVT [ROBOT PUSH] SIT [BOLT UP] RES [DOOR]]
[EVT [ROBOT SLIDE] SIT [ROBOT IN] RES [BOLT]]
[EVT [ROBOT MOVE] SIT [ANY] RES [NOTHING]]]

(c) MARY

MKGOAL - [NONE]

MKSEE - [MARY]

MKACTS - [MOVE PUSH]

MKRULES - [[EVT [ROBOT PUSH] SIT [ANY] RES [DOOR]]
[EVT [ROBOT SLIDE] SIT [ANY] RES [NOTHING]]
[EVT [ROBOT MOVE] SIT [DOOR OPEN] RES [ROBOT]]]

To put (b) and (c) into English:

(b) John's goal is to get in; he can see the positions of himself, the door and the bolt, but not of Mary; he can move or slide the bolt, but not push the door; and he believes that

(i) if you slide the door when the bolt is up, the door changes position.

(ii) if you slide the bolt when you are in, the bolt changes position.

(iii) if you move, nothing happens.

(c) Mary has no goal; she can only see herself, and she cannot slide the bolt. She believes that:

(i) if you push the door, it changes position.

(ii) if you slide the bolt, nothing happens.

(iii) if you move when the door is open, you change position.

In addition to setting these variables, it is also necessary to load routine ZRBASIC in both robots. The variables which hold the control information (akin to CONTROL in 2.5.1) are called JECONTROL and MECONTROL. When JRBASIC has been loaded, the value of JECONTROL is:

```
[[NAME JRBASIC KIND ROUTINE PLACE 1 ENTRIES [2 NIL 4 NIL 6
    NIL 8 NIL]]]
```

This is an association list, and it should be mainly self-explanatory. The entries list corresponds to the instructions marked '*' in the definition of JRBASIC; at present, no entries have been made, so each is set to NIL, which is used in POP-2 to represent the empty list [ ]. To stop John trying to achieve the goal on his own, we preset one of the entries, filling in entry 4 as [FAILED]. This will cause John to ignore the solo attempt and go directly on to the joint attempt, the one involving games.

Having initialised the variables, loaded JRBASIC and

MRBASIC, and made entry 4 in JRBASIC equal to [FAILED], we now call the chairman function, and the run commences. If we ask for a partial print-out we get R1; if we ask for a full print-out, we get R2.

To illustrate how games are loaded and run, we consider the first six utterances. (It will be noted that the utterances on R1 and R2 are numbered; to refer to utterances we will use J or M followed by the number. Thus utterances 1 - 6 are J1, M2, J3, M4, J5, and M6.) Before going ahead with this, it is necessary to say briefly how the top level of the problem works. There are four basic things it can decide to do:

1. CALL the other robot
2. SEND a message to the other robot
3. SWAP control
4. CONTinue with the next instruction in control.

John's functions for doing these four things are JECALL, JESEND, JESWAP and JECONT. Which of the four gets done depends on a variable JENEXT, which can be set to any of the four. Any part of the program can change the setting of JENEXT, and thus determine what the executive does next time control returns to it. If a game wants to get a message sent (i.e. make an utterance) it (a) puts the message

in a variable JEBOX (b) puts JESEND into JENEXT. When JESEND is called, it puts the message in JEBOX into WMESSAGE and exits, control reverting to the chairman. The manoevre of putting a message in the "box" and telling the executive to send it (rather than carry on with games and routines) is called "posting" the message. If a message is posted it will be collected and sent as soon as control goes back to the executive.

We now go step by step through the first six utterances on R2.

The chairman arouses John first, and since John begins with JENEXT sets to JECONT (its normal value), John's executive (henceforth JE) looks in JECONTROL and finds that it is in JRBASIC at place 1. The first function in JRBASIC is thus called, hence the line "1 CALLED BY JOHN" in R2: "1" means the first instruction of the top structure in JECONTROL. Instruction 1 has no entry, and its function, JRPREP, gets the world model ready and then puts up the instruction place to 2. (Instructions without entries have to arrange that the next instruction gets called.) After JRPREP has finished, JECONTROL is:

[[NAME JRBASIC KIND ROUTINE PLACE 2 ENTRIES [2 NIL 4 [FAILED]
6 NIL 8 NIL]]]

Control loops back to the executive, JECONT is called again, and instruction 2 is implemented. This has an entry,

the main goal, hence the line (in R2) "2 [JOHN IN] ENTERED BY JOHN". Once the function in instruction 2 has finished, the list after ENTRIES in JECONTROL is

[2 [JOHN IN] 4 [FAILED] 6 NIL 8 NIL]

So it goes on: 3 is called; 4 is skipped as the entry is already made, 5 is called, and then JE comes to instruction 6, where an appeal has to be made to Mary to find out if she will help with the goal. The game which is used to agree on a joint goal is JGGOAL, so the task is to get this game loaded.

The function which has the job of arranging a given game is called JEPLAY. It has two arguments: the name of the game, and the first move. So the function in JRBASIC 6, JRAPPEAL (see definition of JRBASIC) calls

JEPLAY ("JGGOAL", [JOHN IN])

indicating a desire to suggest the goal [JOHN IN]. JEPLAY proceeds as follows:

(a) it saves the game and the move, by putting them into global variables JEGAME1 and JEMOVE1.

(b) if John and Mary are already co-operating, and no game is in progress, it loads JGGAME with John as White.

(c) otherwise, it arranges for Mary to be called by putting JECALL into JENEXT, then loads JGGAME with John as White.

In this case, (c) applies rather than (b), so JGGAME is loaded (see R2) and when control reverts to the executive, the games and routines are ignored and Mary is called; in other words, JE does JECALL, instead of JECONT as previously. JECALL puts [MARY] into WMESSAGE; puts JECONT into JENEXT, so that next time John is aroused he will go on with his routines and games rather than calling Mary for ever; and exits, control passing to the chairman. The first utterance has been made. JECONTROL is now:

GAME JGGAME Colour: WHITE Place: 1 Entries: [1 NIL 2 NIL]
ROUTINE JRBASIC Place: 6 Entries: [2 [JOHN IN] 4 [FAILED] 6 NIL 8 NIL]

Mary is now aroused. A robot always tests for having been called at top level: if it has been called, MGGAME is loaded without further ado and the robot says [YES] and exits. The reply to a call is not a move in a game, and in fact it has the same effect as swapping: it is ignored. When Mary says "Yes", MECONTROL is:

GAME MGGAME Colour: BLACK Place: 1 Entries: [1 NIL 2 NIL]
ROUTINE MRBASIC Place: 1 Entries: [2 NIL 4 NIL 6 NIL 8 NIL]

John is aroused again: he has not been called, and JENEXT is JECONT, so he goes on with the next instruction in the top structure of JECONTROL, namely, the first instruction of JGGAME. The entry is [JGGOAL], and this is translated into English and posted.

The definition of JGGAME is:

```
GAME JGGAME;
1. * W JGNAME  [JDGAME];
2. * B JGREADY  [JDSIGN];
3. ↑ W JGLOAD;
END;
```

Mary processes John's remark ("I want to suggest a goal") with MDGAME, and finds that the remark is indeed a game suggestion, and translates to [MGGOAL]. This is entered and she goes on to the next instruction. It is a Black instruction, and since she is Black she calls MGREADY. Eventually, this will give John a sign that Mary is willing to play the game.

But she cannot play it yet. To say whether or not she will assist with a goal, she must first note what her own main goal is: she must have made entry 2 in MRBASIC. Since John called her before she got started on MRBASIC, she is still on instruction 1. So the game has to be held up. MGGAME is taken out of MECONTROL and held (in MEHOLD).

Then a mark is made at MRBASIC 3 and ME goes through MRBASIC normally until the mark is reached, whereupon the mark is removed, the game is reloaded, and the reply to John is posted. Before exiting, MGREADY gets ready to play MGGOAL, by ending MGGAME and loading MGGOAL in its stead.

John, of course, is still in JGGAME. He reads Mary's reply, M4, processing it with JDSIGN and finding it acceptable. Going on to JGGAME 3, he calls JGLOAD, which ends JGGAME and loads JGGOAL. When John makes the first move in JGGOAL, the two controls (JECONTROL and MECONTROL) look like this:

John	Mary
GAME JGGOAL Colour: WHITE Place: 1 Entries: [1 [JOHN IN] 2 NIL]	GAME MGGOAL Colour: BLACK Place: 1 Entries: [1 NIL 2 NIL]
ROUTINE JRBASIC Place: 6 Entries: [2 [JOHN IN] 4 FAILED 6 NIL 8 NIL]	ROUTINE MRBASIC Place: 3 Entries: [2 [NONE] 4 NIL 6 NIL 8 NIL]

When Mary has entered John's first move and made her reply, [YES] (or "By all means") John can end JGGOAL and return [YES] to JRBASIC. Having made the entry for JRBASIC 6 he can go on to 7, then 8, whereupon JRACHIEVE is loaded to try to achieve the goal. At this point we conclude the example. In future we shall assume that such expressions

as "loading a game" or "making an entry" can be understood without recourse to diagrams of the state of JECONTROL or MECONTROL, and that all that need be said with regard to R2 is to point out any unusual features that arise: the basic process should be clear from the above account.

2.6.3 Game to arrange games (JGGAME)

In describing games, we shall first give the definition, then explain each instruction, and finally give examples from both the White and the Black points of view.

The definition of JGGAME is:

```
GAME JGGAME;
1. * W JGNAME  [JDGAME];
2. * B JGREADY  [JDSIGN];
3. ↑ W JGLOAD;
END;
```

Now we describe the behaviour of each instruction.

1. * W JGNAME [JDGAME]

When a routine or a game wants to arrange for a game to be played, it used the function JEPLAY, as was mentioned earlier. JEPLAY takes two arguments: the name of the game, which is saved in JEGAME1, and the first move, saved in JEMOVE1. Consequently, all JGNAME has to do is to enter the name it finds in JEGAME1. The program is written so that if John makes an entry in a game, and the instruction

in question is his colour, then the entry is also translated into English and posted. So whenever we talk of an entry being made during the execution of an instruction in a game, it can be assumed that the entry will also be translated and posted: in short, it becomes an utterance. The possible entries for this instruction are: [JGASK], [JGGOAL], [JGTELL] [JGRULE], [JGASSESS], [JGPLAN]. Mary will enter the equivalent versions beginning with M ([MGASK] etc). This is in fact the only case in which the entry John makes is not exactly identical to that subsequently made by Mary. There is also a special entry, [JUMP], which is used to indicate that a muddle has arisen, and which causes the robots to jump out of all routines and games above routine JRBASIC. [JUMP] is translated into English as: "We have got muddled; lets start again". This entry does not appear in R2, and little attention will be paid to it; its purpose is to take the robots out of a game which has gone wrong (i.e. in which someone has said something inappropriate) and start them off some way back in the computation so there is a chance of the difficulty not arising next time.

2. * B JGREADY [JDSIGN]

The answer to White's request that a game be played is always [YES]; but before the reply is sent, Black will have made various preparations to play the game, and will

have loaded it once these preparations are complete. The nature of the preparations depends on the entry made in 1.

(a) If the entry is [JUMP], Black enters the reply [YES] and jumps out of all routines and games above BASIC.

(b) If the entry is [JGGOAL] or [JGPLAN], Black may have to go back to the routine until a certain point is reached. In the case of JGGOAL, Black must be past JRBASIC 2 in order to play, and if not, he marks the routine at instruction 3 and suspends the game until this point is reached. With [JGPLAN], he must be at JRACHIEVE 5 to play the game, so a mark is left at this point. It is possible to leave a mark for a routine not yet loaded: the computation will still go on until the routine in question <u>is</u> loaded, and the right instruction reached, whereupon control reverts to the game.

(c) If the game entered in 1 is [JGTELL] or [JGRULE] - a game to tell Black something, or explain a rule - then if a game was already in progress, it is stripped of all entries and set back to the beginning. The reason is that if, for example, the robots were making a plan, and John interrupted to explain something, it can be assumed that the interruption bears on the game in progress - on the plan-making - and that once the information had been given, the parties will want to

make different moves in the original game; for example, Mary may want to suggest a different plan. So if a conversation is interrupted by JGTELL or JGRULE, it must be restarted from the beginning.

These preparations being made, Black enters [YES], exits from JGGAME, and loads the game entered in 1 in its stead, taking the colour Black in the new game.

3. ↑ W JGLOAD

If his entry in 1 was [JUMP], White jumps back to JRBASIC; if it was a game, he loads this game in place of JGGAME, again taking the colour WHITE.

An example of JGGAME was given in 2.6.2, including the delaying of the game until a mark at JRBASIC 3 was reached. Another example of marking, this time following the suggestion by John of [JGPLAN], is supplied by J7 - M8 in R2. Here there is an extra complication, as while Mary is progressing towards her mark at MRACHIEVE 5, she reaches a point where she wants to ask a question. The effect of this is to disrupt the game in progress (JGGAME is special in that it cannot be interrupted, for to do so would be to play another version of it, which we have not allowed). As a result, when Mary has asked her question (M8 - J13) it is she who arranges the game to choose a plan (M14): the fact that John suggested it at J7 is forgotten by both. It should be

noticed that Mary does not have to call John at M14; since they are co-operating, and the previous game is ended, she can assume that he will take her utterance to be the first move in JGGAME. The same applies to J7, but Mary at M8 is interrupting JGGAME, and thus has to call John in order to get JGGAME loaded (with her as White) and arrange for the asking of a question.

As an example of an interruption which causes the original game to be restarted, consider M18 - J37 (see R1). Mary suggests a plan at M28, making the first move in JGPLAN. John interrupts and arranges JGRULE, which (see JGGAME 2, (c) above) has the effect of removing the entry just made in JGPLAN. Consequently, when JGRULE is finished (at J35) Mary suggests a plan again, although it is the same one (as it happens). Having been persuaded by the discussion from J33 - J35, John agrees to the plan this time, and JGPLAN ends.

2.6.4 Game to obtain information (JGASK)

The definition of JGASK is

```
GAME JGASK;
1. * W JGQUERY  [JDQUERY];
2. * B JGANSWER  [JDSIGN];
3. ↑ W JGRECORD;
END;
```

There are two kinds of question that can be asked with this game, one with the form "Is this the case?", the other "Can you do this?" Examples of "is" questions are:

Is the door open?

Are you out?

and of "can" questions:

Can you move?

Can you slide the bolt?

It will be noticed that all these are Yes-No questions: the answer is always a "sign"; that is, either Yes, No, or Don't Know.

In terms of the internal language, a question is either an event preceded by "Can" or a situation preceded by "Is". Events (as will be recalled from section 2.4) are expressions such as [JOHN PUSH], [MARY SLIDE], [JOHN MOVE], and situations are expressions like [BOLT DOWN], [DOOR OPEN], [MARY OUT]. Thus the expressions corresponding to the English questions above are:

Is the door open?	[IS DOOR OPEN]
Are you out?	[IS MARY OUT]
Can you move?	[CAN MARY MOVE]
Can you slide the bolt?	[CAN MARY SLIDE]

assuming that John is asking the questions. So JDQUERY will try to translate English sentences into expressions of this form, and will return (say) [IS DOOR OPEN] if it

succeeds, and UNDEF if not. As before, a "sign" in the internal language is either [YES], [NO], or [UNDEF], there being a number of positive expressions which count as [YES] and a number of negative ones which count as [NO].

We now give a full account of each instruction. It should be observed that the robots can gain a certain amount of incidental information from the game: for instance, if White asks "Is the door open", Black can infer that White cannot see the door, and update his model of White accordingly.

We will assume that John is White and Mary Black.

1. * W JGQUERY [JDQUERY]

When the game was set up by JEPLAY, the first move was put into JEMOVE1, so JGQUERY merely has to enter whatever lies in JEMOVE1. This automatically causes the entry to be translated and posted (see 2.6.3). Examples of entries were given above; we will follow through the typical questions [IS DOOR OPEN] and [CAN MARY PUSH].

2. * B JGANSWER [JDSIGN]

If the question begins with IS, Black records the fact that White could not see the object; so if entry 1 is [IS DOOR OPEN] Black (Mary) puts a # after DOOR in MKXSEE, her model of what partner can see (see 2.3.3). The answer to the question is then found, ([YES], [NO] or [UNDEF]) and entered.

3. ↑ W JGRECORD

If the question was an IS question ([IS DOOR OPEN]) Black will have replied [YES], [NO] or [UNDEF]. If it is [YES] or [NO], White updates JKWORDD by putting either OPEN or SHUT after DOOR, and also puts a 1 after DOOR in JKXSEE, thus recording that Black could see the DOOR (or so it is assumed). If the answer is [UNDEF], JKWORLD is left unaltered, and a Ø/is put after DOOR in JKXSEE. Next time John wants to find out whether the door is open, he will know that there is no point in asking Mary, and this in fact prevents him from going into a loop when he asks something she doesn't know.

If the question was a CAN question ([CAN MARY PUSH]) it will be answered [YES], or [NO]; Mary must know what she can and cannot do. John puts a 1 after PUSH in JKXACTS if the answer was [YES], and a Ø if it was [NO], JKXACTS being his model of what Mary can do (see 2.3.2).

JGASK does not return a value to the instruction which called it, but changes the robot's memory, his global variables. The result is that when control returns to the instruction which set up JGASK, the information it needed will now be available in the memory. Utterances M22 - J27 in R2 are a case in point. When Mary calls John at M22, her

control structure is:

```
ROUTINE MRPLAN        Place: 4
Entries: [1 [JOHN IN] 2 [EVT [ROBOT MOVE]
  SIT [DOOR OPEN] RES [ROBOT]] 3 [EVT [JOHN
  MOVE] SIT [DOOR OPEN] RES [JOHN]] 4 NIL]
```

```
GAME MRPLAN           Place: 2
Entries:              Colour: White
  [1 NIL 2 NIL]
```

```
ROUTINE MRACHIEVE   Place: 5
Entries: [1 [JOHN IN] 2 [BOTH] 3 [SITN]
  4 [NOTYET] 5 NIL 7 NIL 8 NIL 9 NIL
  10 NIL 11 NIL]
```

```
ROUTINE MRBASIC       Place: 8
Entries: [2 [NONE] 4 NIL 6 NIL 8 NIL]
```

Thus she is trying to agree a plan (MRACHIEVE 5) by playing game MGPLAN, and since she is White and has to suggest a plan, she has loaded routine MRPLAN to find one. Instruction 4 of MRPLAN (see 2.5.4) looks at the SIT mentioned in entry 3, [DOOR OPEN], and enters its state; [ACHIEVED], [FAILED] or [NOTYET]. To find its state, Mary must know whether the door is open, and she examines MKWORLD. But the symbol after DOOR is UNDEF: she doesn't know. She then checks MKXSEE to find out whether John can see the door: she doesn't know this either, so it is worth trying a question

in case he can. Instruction 4 thus sets up game MGASK, using MEPLAY with arguments "MGASK" (the game) and [IS DOOR OPEN] (the first move), and then exits with the entry still unmade.

Since game MGPLAN is already in operation, Mary has to call John to get MGGAME loaded, and get him to play MGASK (M24). When MGASK is over (M26 and J27), John's control structure returns to what it was - he is in JRPLAN and expects Mary to suggest a plan - and Mary is still on instruction 4 of MRPLAN, exactly as she was before M22 (see initial diagram). But there are two changes in her memory: the symbol after DOOR in MKWORLD is now SHUT instead of UNDEF, and the symbol after DOOR in MKXSEE is 1. She has learned that John can see the door and that it is shut. So when instruction 4 is called again, it finds that [DOOR OPEN] is not yet achieved, and enters [NOTYET]. As a result, the goal [DOOR OPEN] is included in the plan which MRPLAN returns to MGPLAN, and hence in the plan Mary suggests as her first move in that game.

In brief then, the process is as follows:

(a) During a routine or game, an instruction consults a global memory variable and finds an UNDEF, indicating ignorance.

(b) It sets up a game to replace the UNDEF with something more informative, then exits without making its entry or

moving control to the next instruction.

(c) The game is played, and the UNDEF, hopefully, is replaced.

(d) The instruction is called again and, since the information is now available, does its job, either making an entry or moving control to a new instruction.

2.6.5 Game to give information (JGTELL).

The definition is:

```
GAME JGTELL;
1. * W JGRELATE [JDFACT JDEVENT];
2. * B JGEXAMINE [JDSIGN];
3. ↑ W JGRELOOK;
END;
```

This game is played on two occasions: first, when John has just carried out an action which is part of a joint plan, and wants to say that he has done so; second, when Mary says or implies something which John believes to be false, and he wants to put her straight. Utterances M42 - J45 in R1 are an example of the first use, and J49 - M54 are an example of the second.

John either relates an event or a fact (an event being the report of an action, a fact a statement about the world). An event is an expression like [JOHN PUSH], and

has already been met; a fact is an IS question preceded by a truth-value. Here are some translations of things John might say at JGTELL 1.

I have pushed the door	[JOHN PUSH]
I have moved	[JOHN MOVE]
The door is open	[1 IS DOOR OPEN]
The door is not shut	[Ø IS DOOR SHUT]
You are not in	[Ø IS MARY IN]
I am out	[1 IS JOHN OUT]

In reply to a statement, the "signs" [YES], [NO], [UNDEF] take on a slightly different significance, though a related one. [YES] means "I already knew that", [NO] means "I disagree", and [UNDEF] means "I accept what you say". These meanings correspond to the answers Black would have given to equivalent questions: for instance, if he would have answered "I don't know" to the question "Is the door open", he says "I see" (meaning "I accept that" and implying that he didn't know it already) in reply to the statement "The door is open". [UNDEF] is thus the internal translation of both "I see" and"I don't know".

The possible outcomes and side-effects of JGTELL are rather complex, as the account of each instruction will show:

1. * W JGRELATE [JDFACT JDEVENT]

The fact or event, put in JEMOVE 1 when John set up the game, is entered, e.g. [Ø IS DOOR OPEN] or [JOHN PUSH].

2. * B JGEXAMINE [JDSIGN];

If entry 1 was an event, Mary (as Black) makes sure she has reached MRACHIEVE 8 before proceeding. This is because it is at 8 that one enters whether the action has been done, and if the game is to return this value to the routine, the right instruction must first have been reached. So she may have to mark MRACHIEVE 8 and hold up the game until the mark is reached. Then she

(a) enters [UNDEF] as MGTELL 2.

(b) exits from MGTELL and enters [DONE] at MRACHIEVE 8.

(c) records that John can do the action he did (e.g. puts 1 after PUSH in MKXACTS)

(d) updates the tree, marking the current goal as ACHIEVED.

If, on the other hand, entry 1 was a fact such as [Ø IS DOOR OPEN], Mary finds out what answer she would have given to [IS DOOR OPEN], (by examining MKWORLD) and whether she can see the door (by examining MKSEE). If she cannot see the door she accepts John's statement, enters [UNDEF], updates MKWORLD and MKXSEE (thus recording what he said and the fact that he knew), and ends the game. If she can see the door, then if John agrees with her she puts 1 after DOOR in MKXSEE (he does know), and if not Ø (he doesn't know). She enters [YES] in the first case ("I know") and [NO] in the second ("I disagree"), and ends the game.

3. ↑ W JGRELOOK

If his entry 1 was an event, John exits from instruction

and game; there is nothing more to be done. If he entered a fact, he examines Mary's reply. If she disagrees ([NO]) with his statement, he finds out whether he can see the object in question. If he can, he records that she can't; and vice versa. So if DOOR is in JKSEE (in the above example) he puts a Ø after DOOR in JKXSEE, but if DOOR is not in JKSEE he puts a 1.

If Mary says "I know" ([YES]) he puts a 1 after DOOR in JKXSEE, but if she says "I see" he puts a Ø. So he uses Mary's answer to update his model of what she can see.

Two brief examples from R2 should suffice:
Utterances M42 - J45 are concerned with loading and playing that form of game JGTELL which relates that an action has been done. Mary has pushed the door, having reached MRACHIEVE 8, and she sets up a game to tell John. No other game is in progress, so there is no need to call him at M42. Her entry at MGTELL 1 is [MARY PUSH]; before answering, he holds up the game until JRACHIEVE 8 is reached, even though this involves a new version of JRACHIEVE (see M44 - J45). John enters [DONE] in JRACHIEVE 8 from the game, but Mary had already entered [DONE] before arranging the game. The result is that John never even calls instruction 8: he finds the entry already made from outside

Utterances M64 - J69 illustrate the other use of JGTELL.

They are playing game JGASSESS (from J61) which is used to assess the consequences of an action - John moved. Mary had found out earlier that John was out (M12 and J13) and, not being able to see him, she continues to believe this although he has now moved through the open door and is in; J63, "I have changed position", implies that John is in; and Mary, believing otherwise, sets up game JGTELL to tell him that he is out. Since M64 interrupts JGASSESS, it must be a call of John's name to get JGGAME loaded. As we said in 2.6.3, a call of JGTELL (or JGRULE) from inside a game causes that game to be stripped of entries and restarted: J63 is thus cancelled. At J69, John finds he can see himself, assumes that Mary can't, and contradicts her. Since in fact she cannot see him, she accepts his version (in MGTELL 3) and updates MKWORLD and MKXSEE accordingly. So she now believes that John is in. As a result, when John restates JGASSESS 1 (at J69, second half) - "I have changed position" - she accepts what he says (M70, M72). In an earlier case, M46- M54, it is John who interrupts JGASSESS, and since he can see the door and Mary can't, she accepts what he tells her, and makes a different MGASSESS 1 (compare M48 with M54 part 2; at M48 "nothing has happened"; at M54/2 "the door has changed position").

2.6.6 Game to discuss rules - JGRULE

There is one example of this game in R1, from J33 - J35.

It arises when Mary has suggested a plan (M28) which, according to John's rules, will not work. He interrupts game JGPLAN (J29, M30) and arranges a game in which he can explain the relevant rule (J31, M32). The ensuing game, JGRULE, has four utterances (J33, M34/1, M34/2, J35) and the format is roughly as follows:

(a) John expounds the rule which he wants Mary to adopt.

(b) Mary compares John's rule with hers. If his is "better" (i.e. more complex) it is accepted and the game ends; if not, she disagrees and makes utterance (c).

(c) Mary expounds her rule.

(d) If Mary's rule is better, John accepts it; if they are equal, John keeps his old rule but records Mary's in his model of her.

In the example, Mary's rule is better than John's since it contains a proviso; as a result, the whole game is played, from (a) to (d), and at the end of it John has not only learned what Mary's rule is, but has also adopted it himself.

The definition of JGRULE is

```
GAME JGRULE;
1. * W JGWRULE [JDRULE];
2. * B JGBREPLY [JDSIGN];
3. ↑ W JGWNOTE;
4. * B JGBRULE [JDRULE];
5. * W JGWREPLY [JDSIGN];
6. ↑ B JGBNOTE;
END;
```

Instructions 1, 2, 4, 5 (those marked *) correspond to utterances (a), (b), (c), (d) above, and to J33, M34/1, M34/2 and J35 in R1.

The utterances of this game are processed either by JDRULE or by JDSIGN. As usual a "sign" is [YES], [NO] or [UNDEF], and the English translation is the same as that used in JGTELL: "I know", "I disagree" or "I see". So if Black (on instruction 2) has the same rule as White, she says "I know" ([YES]); if she has a different, but worse rule, she says "I see" ([UNDEF]); and if she has a different, but equal or better rule, she says "I disagree" ([NO]).

A "rule" is of the form used in JKRULES (see 2.3.2). The rules uttered at J33 and M34/2 are translated by JDRULE into this form:

J33: [EVT [ROBOT MOVE] SIT [ANY] RES [NOTHING]]

"If you move, nothing happens".

M34/2: [EVT [ROBOT MOVE] SIT [DOOR OPEN] RES [ROBOT]]

"If you move when the door is open, you change position".

We now describe the behaviour of each instruction.

1. * W JGWRULE [JDRULE]

Enters the rule in JEMOVE 1. As before, this causes the utterance to be translated and posted.

2. * B JGBREPLY [JDSIGN]

Black first updates her model of White's rules, by

inserting entry 1 into MKXRULES. Then she compares White's rule with her corresponding rule (i.e. her rule for that event). If they are the same, she enters [YES] and exits from the game. If White's is better, she enters [UNDEF], replaces her rule with White's in MKRULES, and exits from the game. Lastly, if her rule is equal to or better than White's, she enters [NO] and stays in the game.

3. ↑ W JGWNOTE

If entry 2 is [NO], White goes to instruction 4, where he will await an utterance from Black. Otherwise, he can assume that Black has adopted his rule (or already knew it), so he updates JKXRULES accordingly, and exits from the game.

4. * B JGBRULE [JDRULE]

A parallel instruction to 1: Black enters her version of White's rule.

5. * W JGWREPLY [JDSIGN]

White's reply is parallel to Black's in instruction 2. If Black played correctly at 2 and 4, White's rule will be either equal to or worse than Black's (as regards complexity) and they will not be identical. Hence White should only use entries [NO] and [UNDEF], the former indicating that the rules are equal in complexity, the latter accepting Black's as better. In any case, Black's rule is inserted in JKXRULES; and if it is accepted ([UNDEF]), it is inserted in JKRULES as well. White exits from the game whatever

his entry.

6. ↑ B JGBNOTE

If entry 5 is [UNDEF], White has accepted entry 4, so Black updates MKXRULES. In any case she exits from the game.

It is worth returning to the example (J33 - J35) again, with two purposes:

(a) to put the call of JGRULE in context

(b) to show how Mary can make two consecutive utterances (M34).

(a) The context of J33-35 (see R2) is the longer sequence M28 - J33. At M28 Mary makes the first move in MGPLAN, but John interrupts the game to explain what he believes is the right rule. The point is that if this rule is correct, it follows that Mary's plan won't work: the rule predicts that her plan will not achieve its goal. The game JGRULE is played, and it emerges that Mary's rule is in fact better, so John accepts it. But if a game is interrupted by JGTELL or JGRULE, the game must be restarted; so on returning to JGPLAN, Mary repeats the first move (M36). Although M36 is the same as M28, John has now abandoned his old rule in favour of Mary's, and he thus works out that the plan will achieve its object, and agrees to it (J37).

(b) Although Mary makes two consecutive utterances at M34, she gives control to John in between them (see R2). Game JGRULE has six instructions in the sequence WBWBWB. After

Mary's utterance M34/1, which is the entry for instruction 2, John carries out instruction 3, which has no entry, and passes on to 4. He finds that this is not his instruction, and that there is no new utterance from Mary to count as its entry, so he swaps control without saying anything. Mary comes to instruction 3, and since it is a White instruction and has no entry she passes over it and comes to 4. This is her instruction, and she makes the entry and hence the utterance (M34/2). So John processes M34/1 and M34/2 separately, at different times, and as different moves in the game.

2.6.7 Game to arrange co-operation on a goal - JGGOAL

This game was the subject of the example in 2.6.2, so we give only its definition and an account of each instruction.

```
GAME JGGOAL;
1. * W JGPLEAD [JDSITN];
2. * B JGREACT [JDSIGN];
3. ↑ W JGREPORT;
END;
```

1. * W JGPLEAD [JDSITN]

Enters the goal in MEMOVE 1. Examples of entries are [DOOR OPEN], [JOHN IN], etc - situations rather than events. An event such as [JOHN PUSH] could be a subgoal, but is never

given as a main goal since (a) it needs no plan, and (b) it cannot be attained jointly.

2. * B JGREACT [JDSIGN]

Black says [YES] if her goal is the same or if she has none, and [NO] if her goal is different. If she agrees to help, she goes to MRBASIC 7 to prepare for the joint attempt on the main goal, which is put into MKGOAL (so she adopts John's goal if she had none before). Having entered her reply, she exits from the game.

3. ↑ W JGREPORT

Enters the reply (entry 2) in JRBASIC 6, having ended the game. (This game can, in fact, only be called from JRBASIC 6). For an example see 2.6.2, and R2, utterances J1 - M6.

2.6.8 Game to agree a plan - JGPLAN

Definition:

```
GAME JGPLAN;
1. * W JGSUGGEST [JDPLAN JDSIGN];
2. * B JGRESPOND [JDSIGN];
3. ↑ W JGRETURN;
END;
```

This game can only be played when the robots are co-operating to achieve a goal and both are at JRACHIEVE 5. Game JGGAME

ensures that the right place in JRACHIEVE has been reached by holding up the game and marking it (see 2.6.3). The first entry can either be a plan, or the sign [NO] meaning that White cannot think of one. Examples of plans are:

(a) [BOTH [DOOR OPEN] JOHN [JOHN MOVE]]

meaning (if uttered by Mary)

"I suggest we get the door open and then you move".

as in utterance M36 (see R1).

(b) [MARY [MARY PUSH]]

meaning "I suggest I push the door" (see M40).

In other words, the entry for a plan in JGPLAN is in line with the representation for plans used in routine JRPLAN (2.5.4) and on the tree (2.4.2, 2.4.3).

The sign in instruction 2 can be [YES] or [NO], meaning "I agree" (or "All right") and "I disagree". But if Black disagrees, he does so by explaining the rule on which his disagreement is based, though White could understand a straight "No" if it were given.

The behaviour of each instruction is now given:

1. * W JGSUGGEST [JDPLAN JDSIGN]

White loads the routine JRPLAN, which makes the entry. Usually JRPLAN also asks questions as it tries to build a plan. The entry will be a plan, or [NO], which is translated as "I can't think of one". If the game JRPLAN is interrupted by JGRULE, the entry will be wiped out and this

instruction will be run again. The new entry might be different, as JGRULE can alter JKRULES, and JRPLAN consults JKRULES during its operation.

2. * B JGRESPOND [JDSIGN]

Although there are only three possible entries here, they have different significances depending on whether entry 1 is [NO] or a plan. If it is [NO], indicating that White cannot find a plan, then Black carries out a test to determine whether she will be able to. If so, she replies [YES], meaning "I will suggest a plan, then"; if not, [NO] meaning "Neither can I". In the former case Black reloads MGPLAN and takes White; in the latter case the game ends and returns [NO] to MRACHIEVE 5.

If, on the other hand, entry 1 is a plan, the plan is judged. It is accepted wither if Black thinks it will work, or if Black can't think of another, or if Black knows that it is based on a rule equally as "good" as hers. Otherwise it is rejected. To accept it, Black enters [YES]; to reject it she sets up game MGRULE to explain to White the rule by which she rejects it. If the plan is accepted it is returned to MRACHIEVE 5 and the game ends.

3. ↑ W JGRETURN

If entry 1 is [NO], then if entry 2 is [YES] White reloads JGPLAN taking Black; if entry 2 is [NO], [NO] is returned to JRACHIEVE 5. If entry 1 is a plan, it is

returned to JRACHIEVE 5 if entry 2 is [YES], or [NO] is returned if entry 2 is [NO].

A straightforward example of this game is provided by utterances M38 - J41 on R2. Special features of the game are:

(a) Mary arranges the game when she is at MRACHIEVE 5.

(b) John waits until he reaches JRACHIEVE 5 before playing.

(c) The first move in MGPLAN (M40) is entered by routine MRPLAN.

(d) The game returns a value to JRACHIEVE 5; it does not update global variables as do JGASK, JGTELL or JGRULE.

The example M38 - J41 shows these features clearly and without complications: M14 - J37 is another example of JGPLAN which includes almost all the complications that can be imagined: all sort of games are played on top of JGPLAN.

2.6.9 Game to assess the result of an action - JGASSESS

```
GAME JGASSESS;
1. * W JGCHANGE [JDOBJ];
2. * B JGCONFIRM [JDSIGN];
3. * W JGPARENT [JDFACT];
4. * B JGBENTER [JDSIGN];
5. ↑ W JGWENTER;
END;
```

An example of this game is R1, utterances J69/2 - M72.

It is used to agree on the entries for JRACHIEVE 9 and 10, (the result of the action, and consequent state of the parent goal). When it is played, one robot will have just carried out an action and told his partner that he has done so (J57 - M60). So both will be at JRACHIEVE 8 or 9, and the entry for 8 will have been made. When game JGASSESS is ended, the entries for 9 and 10 will also have been made.

Entry 1 in JGASSESS is processed by JDOBJ, which returns either [JOHN], [MARY], [BOLT], [DOOR], or [NOTHING]; in short, the name of the object which moved, or [NOTHING] if no change was observed. Entry 3 involves the parent goal; if it is [JOHN IN], then entry 3 will be [1 IS JOHN IN] (indicating that the goal is achieved) or [ø IS JOHN IN] (indicating that its state is [NOTYET]). The relevant entries in JRACHIEVE 9 and 10 are made at the end of the game.

The instructions behave as follows:

1. * W JGCHANGE [JDOBJ]

White looks at the world (updating JKWORLD) and compares it to the "world" saved in JRACHIEVE 7, before the action (at 8). If there is a difference, the name of the object which changed is entered; if not, [NOTHING] is entered. It may be that White will make a mistake, for if he has been told the position of an object he cannot see, he will assume it is still where it was, his look having told him nothing

about the change.

2. * B JGCONFIRM [JDSIGN]

Black also looks again at the world and sees if it has changed. If she agrees with White, she confirms his entry by [YES]; if not, she tells him the position of the object in question, arranging game JGTELL for this purpose. The call of JGTELL will erase all entries so far made in JGASSESS, so that White will have to make entry 1 again, perhaps differently.

3. * W JGPARENT [JDFACT]

Constructs a statement about the state of the goal the action was meant to achieve, and enters it, e.g. [1 IS JOHN IN] ("John is now in") or [Ø IS JOHN IN] ("John is not yet in"), if the action was (probably) [JOHN MOVE].

4. * B JGWENTER [JDSIGN]

If entry 3 is thought to be wrong, Black arranges a game to tell White how things really are. Otherwise, Black finds the appropriate entries for MRACHIEVE 9 and 10 (by using the entries MGASSESS 1 and 3), ends the game, and makes the entries.

5. ↑ W JGWENTER

White prepares and makes the entries for JRACHIEVE 9 and 10 as Black did in 4.

Utterances M46 - J57 provide examples of most of the above

points. Note that:

(a) Mary sets up MGASSESS from MRACHIEVE 9.

(b) Since she is blind, she makes a mistake at M48: she was told at M26 - J27 that the door was shut; now she has pushed it open, but cannot see it, so assumes it is still shut. John, who can see it, interrupts to tell her it is now open.

(c) The interruption erased entry 1 of the game, and Mary makes it again (M54); but now it is different, since she knows the door has changed position.

(d) M56 looks repetitive, as in this case (as in most cases) the report of which object moved relates closely to whether the goal was achieved. If the program was run with a completely false set of initial beliefs (in JKRULES) this would not be the case (e.g. if they fried to open the door by sliding the bolt). This rather silly repetition demonstrates the point at which the system becomes limited: it knows when to start a conversation and how to conduct it, but doesn't know the purpose of each utterance.

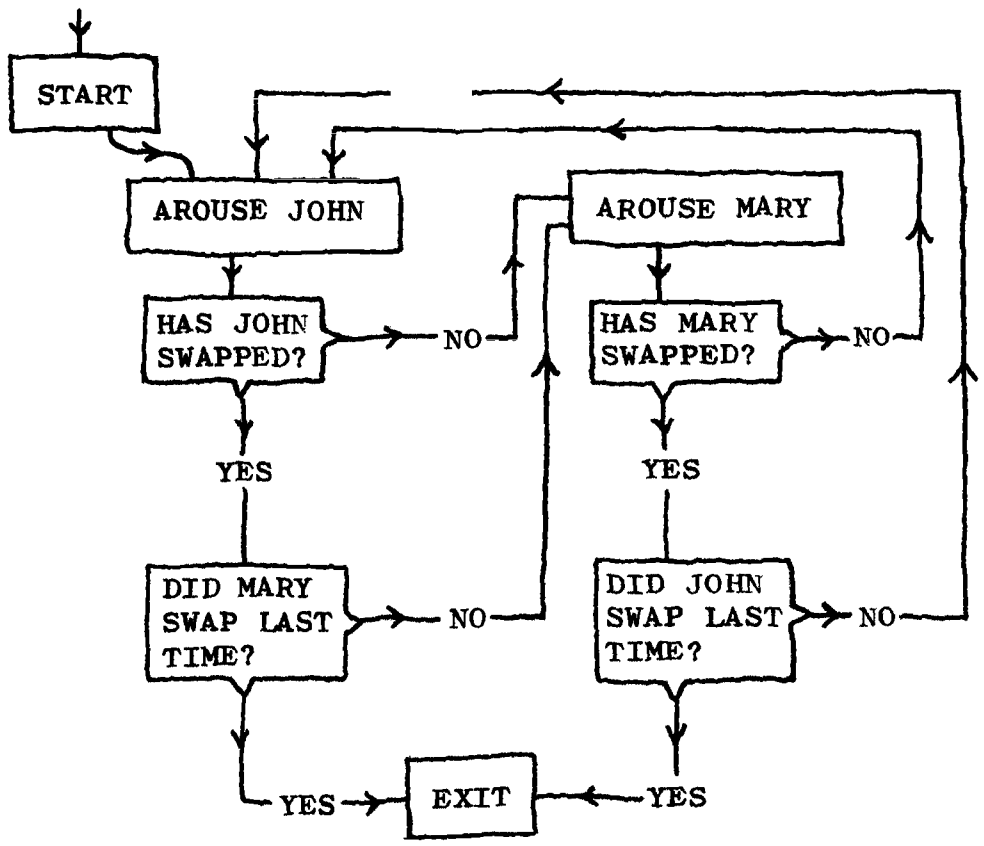
(e) The game (JGASSESS) ends itself and then makes entries at JRACHIEVE 9 and 10 ([DOOR] and [ACHIEVED]).

2.7 Executive

Although routines and games are important control structures, they are not the highest level of control. They

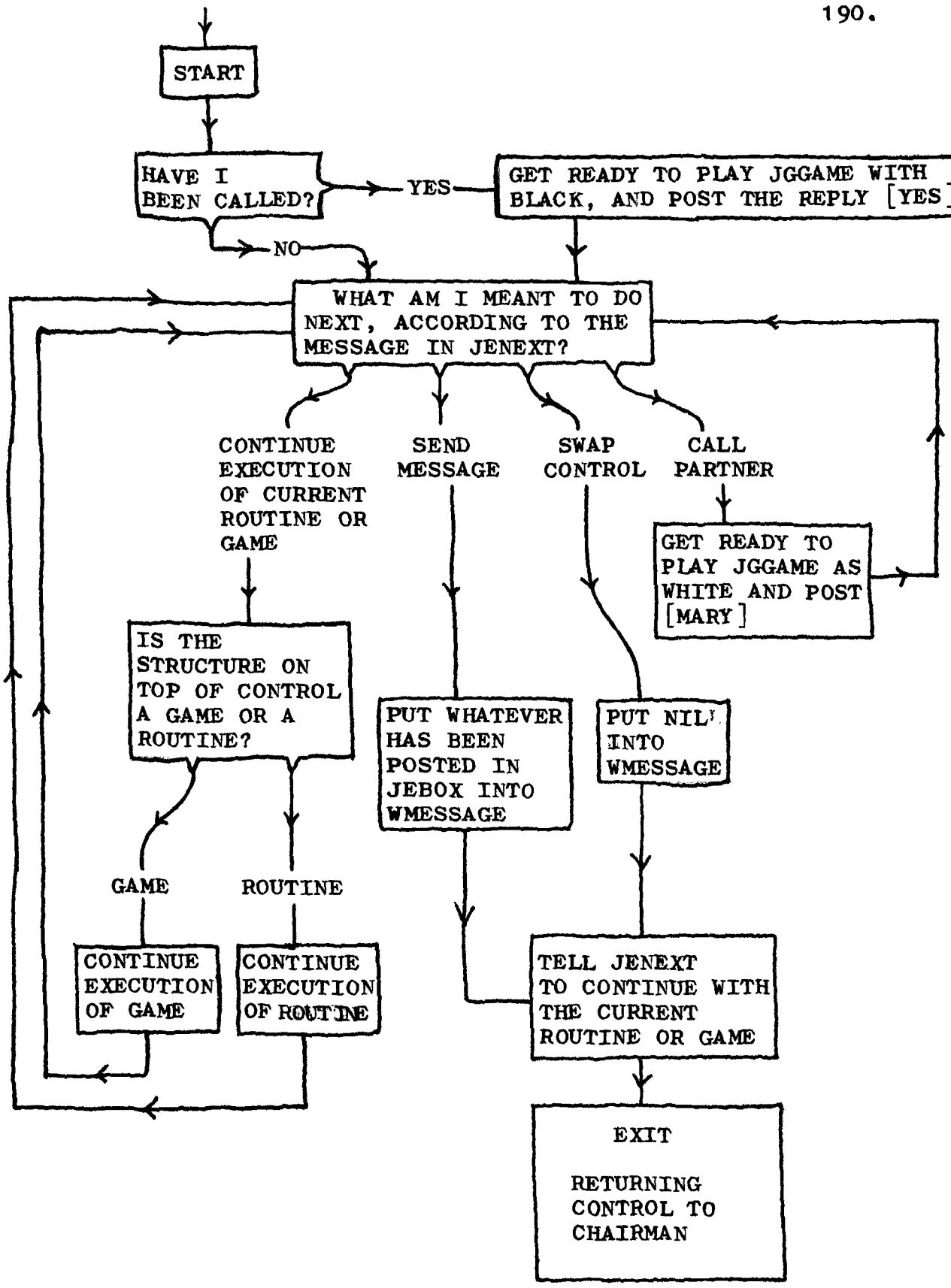
have to be interpreted, and sometimes a decision must be made to discontinue them in order to make an utterance or swap control. The various parts of the program which are above the routines and games are the ones we are now concerned with, under the broad title "Executive". They will be described from the top down, starting with the Chairman, and ending with the interpreters for games and routines. We will rely heavily on flow-chart diagrams as a means of description, and simplify them as much as possible to focus on essentials.

The Chairman program looks like this:



The test for whether John (or Mary) has swapped is made by examining WMESSAGE; if it is NIL, John must have returned and this is how we define "swapping". The test for whether John swapped last time he was aroused is made by examining a local variable which records whether or not John put NIL into WMESSAGE on his last period of arousal.

Thus the only non-trivial part of the algorithm is the box labelled "Arouse John" (or "Arouse Mary"). This represents the revival of John's mind from its dormant state; or in programming terms, a call of JEAROUSE, the master function for the part of the program which represents John's mental processes. When he is aroused, the algorithm representing his mental processes is set in motion, and runs until he makes an utterance or swaps. This algorithm is diagrammed below; it is equivalent exactly to the box "Arouse John" in the Chairman algorithm.


START
HAVE I BEEN CALLED?
YES
GET READY TO PLAY JGGAME WITH BLACK, AND POST THE REPLY [YES]
NO
WHAT AM I MEANT TO DO NEXT, ACCORDING TO THE MESSAGE IN JENEXT?
CONTINUE EXECUTION OF CURRENT ROUTINE OR GAME
SEND MESSAGE
SWAP CONTROL
CALL PARTNER
GET READY TO PLAY JGGAME AS WHITE AND POST [MARY]
IS THE STRUCTURE ON TOP OF CONTROL A GAME OR A ROUTINE?
PUT WHATEVER HAS BEEN POSTED IN JEBOX INTO WMESSAGE
PUT NIL INTO WMESSAGE
GAME
ROUTINE
CONTINUE EXECUTION OF GAME
CONTINUE EXECUTION OF ROUTINE
TELL JENEXT TO CONTINUE WITH THE CURRENT ROUTINE OR GAME
EXIT
RETURNING CONTROL TO CHAIRMAN

Most of these boxes will only need a word or two by way of explanation.

(a) "Have I been called"

This is simply a matter of examining WMESSAGE and seeing whether it says [JOHN].

(b) "Get ready to play JGGAME with Black and post[YES]"

Loading a game has already been described (2.6.1, 2.6.2) - it involves putting a structure on top of the pile of control structures in JECONTROL. To post [YES] involves two operations: first, put [YES] into JEBOX; second, put [.JESEND] into JENEXT. Thus to post a message you put it in the box, and tell JENEXT that it is to send a message.

(c) "What am I to do next, according to the message in JENEXT?

This is the central decision. There are four main things the program can do: continue executing procedures, send a message, swap control, or arrange to call the partner. Usually it does the first of these, but in the course of executing routines and games, the variable JENEXT can be altered so as to get one of the other three jobs done next time control loops back to this decision point. There are four lists which can be put into JENEXT: [.JECONT], [.JESEND], [.JESWAP], [.JECALL], corresponding to the four tasks mentioned above. If, for example, a message has been posted (see (b)), [.JESEND] will be the value of JENEXT, and when this list is evaluated (it contains a piece of program, the call of function JESEND) the value of JEBOX will be "uttered" and JEAROUSE will exit.

(d) "Is the structure on top of control a game or a routine?"

A simple decision, made by inspecting the head of JECONTROL.

(e) "Put whatever has been posted in JEBOX into WMESSAGE" and "Put NIL into WMESSAGE"

are self-explanatory.

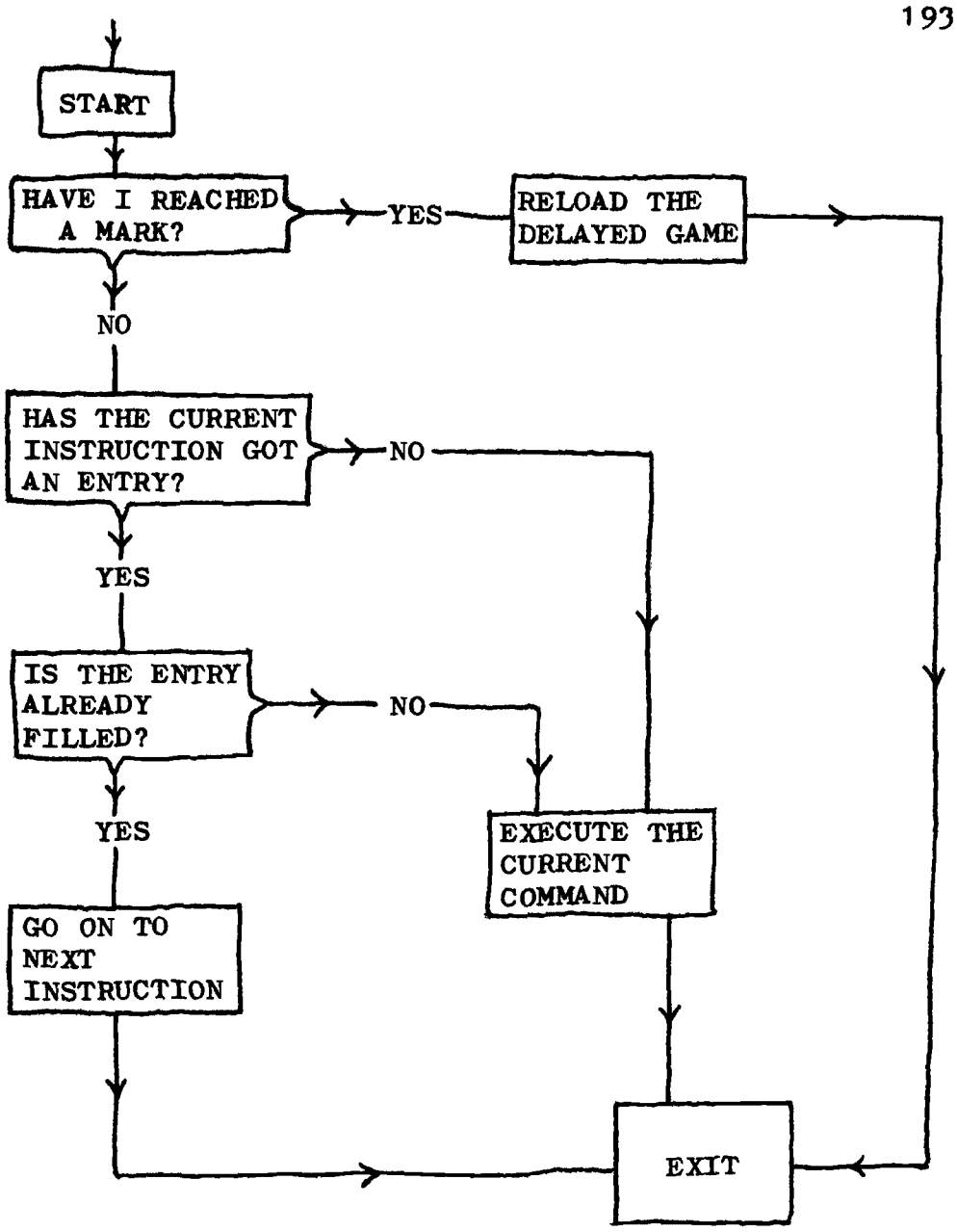
(f) "Tell JENEXT to continue with current routine or game"

In other words, put [.JECONT] into JENEXT. When JEAROUSE is next called by the Chairman, we want it to go on with the routines and games, and not to swap again or to send the same message.

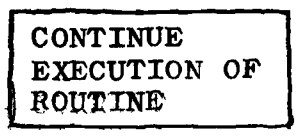
(g) "Get ready to play JGGAME as White, and post [MARY]"

Similar to (b). Calling is thus a special way of sending a message, as is responding to a call as in (b). The more usual way of sending messages is, of course, from games. (By "sending a message" here we mean making an utterance by means of JESEND).

The other two boxes, which continue the execution of the current game or routine, are the interpreters for games and routines, and they merit their own diagrams. We begin with routines, which are rather easier to interpret since they are less complicated. The diagram below corresponds to the box "Continue execution of routine" in the last flow chart.



The above diagram is equivalent to:



The boxes are by now all simple enough to be described verbally:

(a) "Have I reached a mark?"

To make a mark at, say, JRACHIEVE 5, one puts [JRACHIEVE 5] into JEPLACE and the current game into JEHOLD, removing it from the top of JECONTROL. To test for whether a mark is reached, one simply finds out (i) whether any place is marked (if not, JEPLACE will be UNDEF) (ii) if so, is the place in the current routine (iii) if so, am I at that place now.

(b) "Reload the delayed game"

Put UNDEF into JEPLACE (removing the mark) and put the game stored in JEHOLD back on top of JECONTROL.

(c) "Has the current instruction got an entry?"

If the routine is, say, JRACHIEVE, and the place reached is 3, look up the instruction headed 3 in the list holding the definition of JRACHIEVE (i.e. the list held in JRACHIEVE). It will be [3 * JRKIND]. If the second symbol is * there is an entry; if it is ↑, there is none.

(d) "Is the entry already filled?"

If entry 3 is NIL it is empty; otherwise, it is already filled.

(e) "Go on to next instruction"

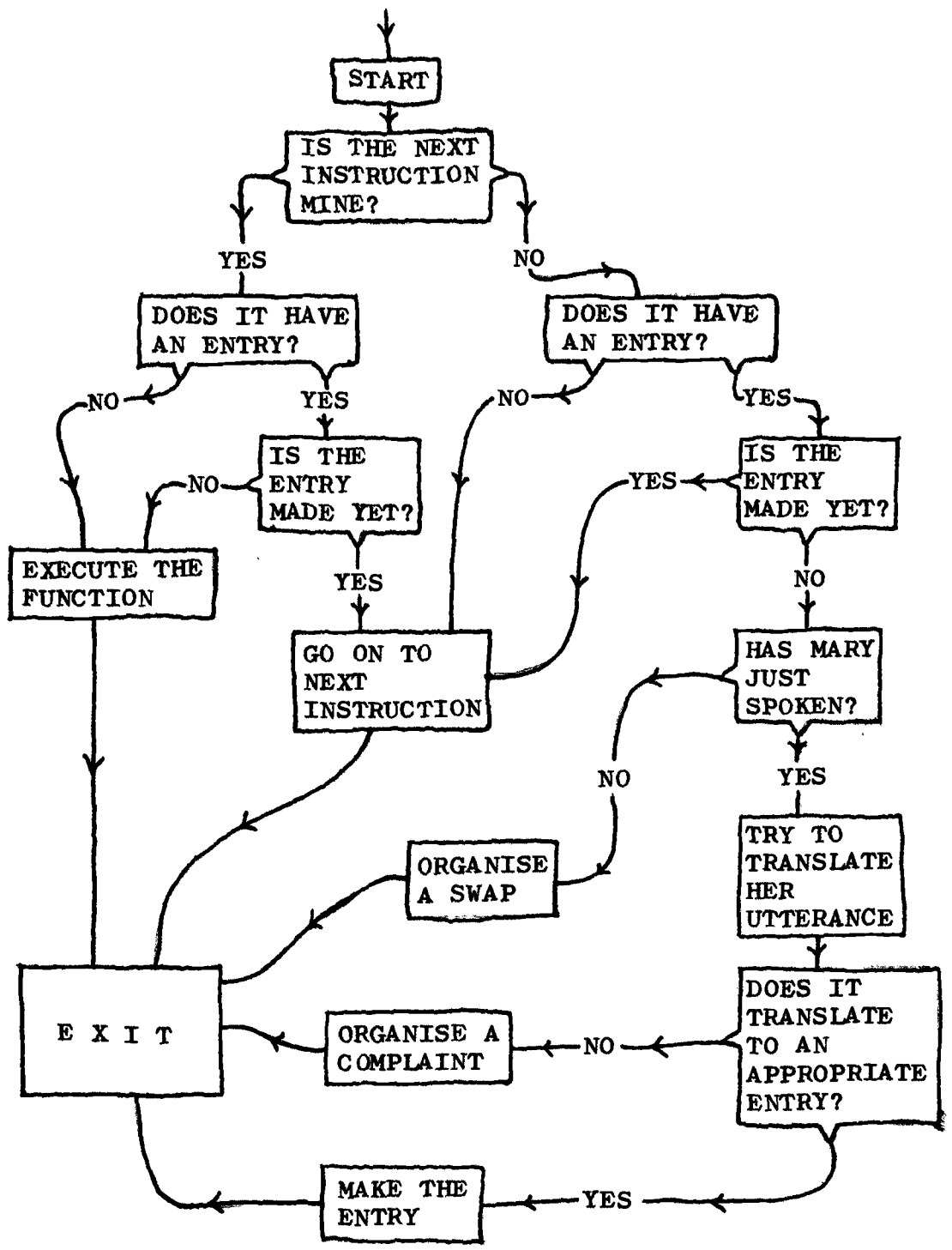
Raise the place you are at by one. Thus if you are at 3, change your place in the routine to 4.

(f) "Execute the current command"

Run the function given in the definition. For

JRACHIEVE 3 this will be JRKIND, as the third instruction is held on the list [3 * JRKIND] (see (c)). It is this function which carries out the task associated with the instruction: these tasks have already been described in section 2.5.

Finally, we give the diagram for the interpretation of games, which was represented in the "Arouse John" flow-chart as "Continue execution of game".



The above diagram is an expansion of box

CONTINUE EXECUTION OF GAME

in "Arouse John" diagram.

Some of these boxes are identical, and some obvious:

(a) "Is the next instruction mine"

Compare own colour with colour of instruction.

(b) "Does it (this instruction) have an entry"

See equivalent box in routine interpretion; same for "Is entry made yet", and "Go on to next instruction", and "Execute the function".

(c) "Has Mary just spoken"

If WMESSAGE is NIL, she hasn't; otherwise, she has.

(d) "Organise a swap"

Put [.JESWAP] into JENEXT.

(e) "Try to translate her utterance"

Apply the list of functions at the end of each * instruction in the game definition (e.g. for a game JGPLAN 1, [JDPLAN JDSIGN]). If a function returns a list, use this list as the entry; if it returns UNDEF, try the next function. If all functions (e.g. both JDPLAN and JDSIGN) return UNDEF, there is no appropriate entry to be formed from the utterance.

(f) "Does it translate to an appropriate entry"

See (e). If it is not UNDEF, it is OK.

(g) "Make the entry"

Obvious: enter it as usual.

(h) "Organise a complaint"

Set up the game JGGAME with the initial move [JUMP].

This translates to "We have got muddled; let's start again".

2.8 Language

In the section on games (2.6) we gave the internal form of each of the English expressions that the system understands (e.g. we said that [Ø IS MARY IN] is the internal form of the statement "Mary is not in"). We have not said how the translation from internal language to English and back is made, apart from mentioning that it is unprincipled. Since the sole purpose of the translation was to make the conversation intelligible to English observers, no effort was made to do a proper parsing, or even to bother with grammar at all. In this section we give an example of the translation methods for one game, JGASK; there is no point in doing more.

JGASK has three instructions, two of which have entries (and hence lead to utterances). These are the question and the answer. We will first consider how a question or answer in the internal language is translated into English.

The functions which construct the entries for instructions 1 and 2 of JGASK are JGQUERY and JGANSWER. To obtain the function which translates the entry into English, the program alters the second letter to W: JWQUERY and JWANSWER. Thus there is a translator function for every move in every game.

JWQUERY takes as its argument a query in the internal

language, and outputs its English translation, held in a list. Examples of the input to the function (see 2.6.4) are:

[IS JOHN IN] [IS BOLT DOWN]

[CAN MARY PUSH] [CAN MARY MOVE]

JWQUERY has to do three things:

(i) Put in pronouns for JOHN and MARY, and adjust the verb IS appropriately (e.g. to "AM" if the pronoun is I).

(ii) Put in a THE before every occurrence of BOLT or DOOR.

(iii) Put in THE DOOR after PUSH and THE BOLT after SLIDE.

As to the answer, JWANSWER can receive as input [YES], [NO], or [UNDEF]. The first two are all right as they stand, so all that JWANSWER does is to alter [UNDEF] to [I DONT KNOW].

To translate English into the internal language, the functions at the end of each line in the game definition are used. Thus for instruction 1 of JGASK, JDQUERY is used, and for instruction 2, JDSIGN.

JDQUERY returns either a translation of the English sentence, or UNDEF, indicating that the sentence cannot be construed as a query. It first subjects the English version to two preliminary functions, one which replaces variations of words like IS or PUSH by the standard versions,

and one which replaces pronouns by robot names. Thus if the English sentence is [I AM PUSHED], the first function changes it to [I IS PUSH] and the second changes it again to [MARY IS PUSH] (assuming Mary uttered it and John is translating). (This is not, of course, anything the system would actually say). JDQUERY then checks that the list is long enough to be a query - at least three words long - and that it starts with IS or CAN. If it begins with IS, the rest of the list is handed to JDSITN, which tries to construe it as a situation such as [DOOR OPEN]; if it begins with CAN, on the other hand, the rest is handed to JDEVENT, which tries to construe it as an event, e.g. [JOHN PUSH].

JDSITN and JDEVENT work by looking for the kinds of word they want. For instance, JDSITN first looks for an object (JOHN, MARY, DOOR or BOLT), and returns UNDEF if it fails to find one. If it gets one (e.g. JOHN) it then looks for an appropriate property (IN or OUT). Should it find one, it returns the completed situation ([JOHN IN], say); if not, UNDEF. Thus JDSITN will translate [GET OUT OF THIS HOUSE, JOHN] or [JOHN IS OUT OF HIS MIND] or [IS JOHN OUT YET] as [JOHN OUT].

If JDQUERY manages to find an IS followed by a situation, or a CAN followed by an event, it gives a translation - [IS JOHN OUT] or [CAN JOHN PUSH]. Otherwise, it returns UNDEF.

JDSIGN has to construe the input as a sign: [YES], [NO] or [UNDEF]. It first tests to see if the English expression is [I SEE] or contains [DONT KNOW], in which case it construes it as [UNDEF] (not UNDEF). If neither phrase appears, it tries to construe it as [YES] by searching for positive words (YES, AGREE, CAN, WILL, GOOD, RIGHT, FINE, SPLENDID, OK, GO, BY, KNOW). If no positive words can be found, it looks for negative ones (NO, DISAGREE, CANT, WONT, BAD, LOUSY, FAULTY, SILLY). And if none of these words or phrases can be found in the expression, UNDEF is returned.

Thus examples of input and output are:

INPUT	OUTPUT
[I SEE]	[UNDEF]
[I REALLY DONT KNOW]	[UNDEF]
[GO AHEAD]	[YES]
[BY ALL MEANS]	[YES]
[I AGREE]	[YES]
[GOOD IDEA]	[YES]
[ALL RIGHT]	[YES]
[I ALREADY KNOW THAT]	[YES]
[I DISAGREE]	[NO]
[THATS A LOUSY IDEA]	[NO]
[I CANT THINK OF ONE]	[NO]

[I WONT HELP YOU]	[NO]
[I DONT AGREE]	[YES]
[GO TO HELL]	[YES]
[I CANT FAULT THAT SUGGESTION]	[NO]

(The last three examples show how silly the translator really is).

The general idea is thus that you try to construct an expression of the required kind (query, sign, or whatever) by searching the input expression for key words, returning either an acceptable construction, or UNDEF if the necessary materials were not found in the input.

3. DISCUSSION

3.1 Overview of the Model

In the next three sections we look at the method by which the program organises conversation, first outlining its main features (3.1), then examining them critically (3.2), and then considering how the model could be improved to meet the criticisms (3.3). Sections 3.4 and 3.5 are devoted to two additional problems: (a) whether the model could readily be adapted so that it was able to learn new conversational facilities (i.e. new games); and (b) whether it could be adapted so that the minds operated in parallel and the chairman procedure was not required. But we begin by summarising the central ideas of the model.

If one is given a piece of ordinary conversation, it is possible to divide it up into natural sections and to assign a purpose to each section. As a rule, these sections will contain two or more utterances, the most common number probably being two. If utterances within a section are shuffled, the result will almost certainly be chaos; for instance, it is essential that questions should precede answers and not follow them. But if whole sections are shuffled, the result will not be so bad: it may appear strange, but is unlikely to be total nonsense.

Thus organisation within sections is much stricter than organisation between them. In our model, organisation between sections derives from two sources, the routines of John and the routines of Mary, and reflects the short-term purposes of both; but organisation within a section arises from a single game, which is run jointly by the robots, and which lays down very strictly what can be said and by whom.

The most important feature of the program, then, is that the conversation is divided into short sections, each of which is generated by a joint procedure. In many cases, the computation done by a game is equivalent to one which might be done privately. The game ZGASK enables John (say) to access Mary's memory, but a similar computation occurs when John accesses his own memory. Again, the game ZGPLAN causes a plan to be generated and then evaluated, but this is also a computation which might be done by a single individual. When these computations are carried out jointly, the processes are basically similar, but are divided up between the parties. The difficulty, of course, is that there cannot be a single master-copy of the game to which both robots can refer: there have to be two copies, one in each mind, and the robots have to keep their copies in line by means of the conversation. Entries have to be coded in English, uttered, and decoded; and if

a robot wants to start a new game, there must be some way in which his partner can find out which game he wants to play.

When a game is in progress, it makes various changes in minds of the robots. Some of these changes constitute the game's chief purpose, while others are incidental side-effects. If, for example, John asks Mary whether the door is open, and she says it is, then the central change is the updating of John's model of object positions, but there are two side-effects: Mary learns that John cannot see the door, while John learns that Mary can. There are three general changes that a game can bring about in a robot: it can alter the world model, send a value to the routine which called it, or alter the control structure underneath it (e.g. by moving to a new instruction in a routine, or killing a routine altogether). Routines have an additional facility - they can cause changes in the world; (though there is no reason why games should not be written to do this as well - for example, one could write a game for making commands to the other robot). The overall effect of games and routines as a whole is to alter the world and the world model, and when a run is completed, and the robots have reached a halt state, it is these alterations which persist.

At the end of section 1.1, we listed some general phenomena which a model of conversation needs to account for. The idea that a conversation is a series of sections

each of which is run by a joint procedure enables us to explain many of these phenomena. For example, a game definition serves the following purposes: it allows a robot to remember where he is by storing the name of the game and the instruction number reached; it provides spaces for entries so that previous utterances can be rememebered; it tells the robot when he should speak and when he should allow his partner to speak; it specifies the range of appropriate utterances at each ppint; and it provides functions for producing utterances, decoding them, and reacting to them. The use of games also makes it very easy to integrate conversations with private thoughts, and to nest conversations inside each other. All these facilities are simply treated as normal cases of one procedure calling another: for a private thought to give rise to a conversation, a routine calls a game; for a conversation to be nested inside another, a game calls a game; and if your partner wants to conduct a conversation unrelated to your private thoughts, all you have to do is to load the relevant game as if your current routine had called it, so that when the game is over, control returns to the right place.

3.2 Criticisms of the Model

There are a number of respects in which the conversations

produced by the model differ from those one would expect from human participants. In this section we examine a number of these defects in the model, and consider whether a system using games could reasonably avoid them. The criticisms to be discussed are the following:

(a) The conversational pattern is too rigid.

(b) John and Mary are too alike.

(c) All sections of conversation have to be announced.

(d) There are no proper facilities for putting right misunderstandings.

(e) The robots cannot detect violations of illocutionary rules.

(f) The robots cannot refer to previous sections of the dialogue.

(g) The robots cannot tell whether or not a game is appropriate.

(a) <u>The conversational pattern is too rigid</u> - there is one game definition for each type of conversation, so that every time a question is asked, or a rule explained, or a plan agreed on, the dialogue takes the same pattern as it did last time. Except for a call of a robot's name, every utterance is expected, in the sense that the receiver knows in advance which move it is in which game. This kind of rigidity is an inevitable result of using games, since

the whole idea of a game is that the robots are following a procedure with a fixed definition which they both know in advance. A certain amount of variety is introduced by the fact that some games end early (e.g. ZGRULE can last for four utterances or for two), and the possibility of interruptions occurring while a game is in progress, so that a new game is nested inside it.

(b) John and Mary are too alike - the method using games depends on John and Mary having exactly equivalent game definitions, and agreeing exactly about the position reached in the game at every point. This seems highly unlikely and unnatural: it means that when the robots talk to each other, they never have misunderstandings, but when a robot talks to an outsider (the operator, say), it will constantly fail to interpret the operator's remarks properly unless the operator is aware of and able to follow the game definitions. If a "naive" human operator is introduced to the program, he will eventually adapt to the program's conversational customs and learn to work with it, but the program is not able to adapt to its partner in the smallest degree. Again, if the program says something unexpected, the operator is still able to make sense of it, but if the operator says something unexpected, the program can do nothing except to notice that it is inappropriate and to complain.

(c) <u>All sections of conversation have to be announced</u> - this objection relates to the previous two. Every time an ordinary game is played, it must first be announced by using the special game ZGGAME. This practise is designed to bring the control structures of John and Mary into line to make sure they have the same game loaded - and it is one of the most unnatural features of the conversation. If a human language user wants to ask a question, he simply asks it, unless he has some special reason for thinking that it might be misinterpreted. It would not be difficult to adapt the program so that extensive use of ZGGAME was unnecessary. The game definitions could be kept as they are, the only new feature being a facility for analysing a given utterance and determining which game, if any, it was meant to introduce. This facility was left out of the program because we wanted the conversational output to reveal clearly what was going on, even at the cost of unnaturalness. A question in English has a dual function: it indicates which game is to be played, and it makes the first move in the game; similarly, an answer to a question also has a double significance: it not only constitutes the second move of the game, but indicates a willingness to play it. The program makes these distinctions clearly, but the overuse of ZGGAME is admittedly tiresome, and is the main barrier to fluent communication with a human operator.

(d) <u>There are no proper facilities for putting right misunderstandings</u> - if a robot cannot interpret a remark of his partner's, he has only one possible response: he calls his partner to get JGGAME loaded, declares that a muddle has arisen, and thus cancels all routines except ZRBASIC and gets a fresh start. This is a most unsatisfactory response, particularly if the misunderstanding is a small one which could be cleared up without going so far back. Human language users are strikingly good at conducting meta-conversations to put previous conversations back on the rails. Within the framework of the model, the problem is to locate the exact place where the mishap occurred and to return control to that place in both minds, cancelling all the entries and procedure calls which followed it. In addition, something must be done to make sure the mistake is not repeated: a game must be called more explicitly, perhaps.

In its present form, the program avoids mistakes by using identical game definitions and calling games explicitly. If it were adapted so that game definitions varied between the robots, and ZGGAME was avoided by using a facility of the kind mentioned in (c), then one would get two kinds of misunderstanding. The first, due to different game definitions, would generally be detected immediately, but there is not much that could be done about it. This is

because the robots have no understanding of why games take the form they do, and hence no ability to modify them. The game is just a list of instructions to be carried out in the prescribed order, and there is no way of giving the robots the ability to modify their games, and thus to bring the games into line, without completely changing the nature of the program.

The second kind of misunderstanding, the one caused by the robots loading different games, might escape immediate detection. For example, if John says "Will you help me get in" (the first move of ZGGOAL), Mary might interpret this as "Are you in?" (the first move of ZGASK) because of the key words YOU IN. If she replies "YES", John will assume she is willing to help, since this happens to be a proper reply to his remark. One way of putting things right would be to have a special game which could be called when the error was detected, and would have the job of rearranging the control structures beneath it so that the robots were brought back into line. The simplest way of doing this is to have one robot editing the control structure of the other. This requires three abilities: to infer that something is wrong, to find out the exact fault, and to rectify it. These are not easy problems: to infer that something is wrong, for example, it is necessary to have some representation of the state which the other mind ought to be in, and to be

able to infer the state which it <u>is</u> in from various behavioural clues. So if Mary carries out an action not mentioned in the plan, John must be able to infer that her planning tree is wrong. Or if John asks for help with a goal, is refused, and then suggests that they choose a plan, Mary must infer that he is (wrongly) executing the instruction ZRBASIC 8 - i.e. he is trying to get the goal achieved jointly. Since the robots lack models of each other's control structures, they cannot be readily adapted so that this kind of task is possible.

(e) <u>The robots cannot detect violations of illocutionary rules</u> - this point was touched on in section 1.5.2. The rules for questions are a good example: if John asks Mary if the door is open, then for this to constitute an illocutionary act, he must really want to know the answer, and not yet know it. If it is obvious to Mary that one of these rules is broken, she should react by pointing this out rather than by answering his question. So if John asks the question, gets his answer, and then asks the question again, the second question should receive a reply such as "I've just told you" or "you already know". Again, if John asks a question which he could have no conceivable motive for, Mary should reply "You don't really want to know that" or "Why do you want to know that?". The ability to give these replies requires that Mary should know a

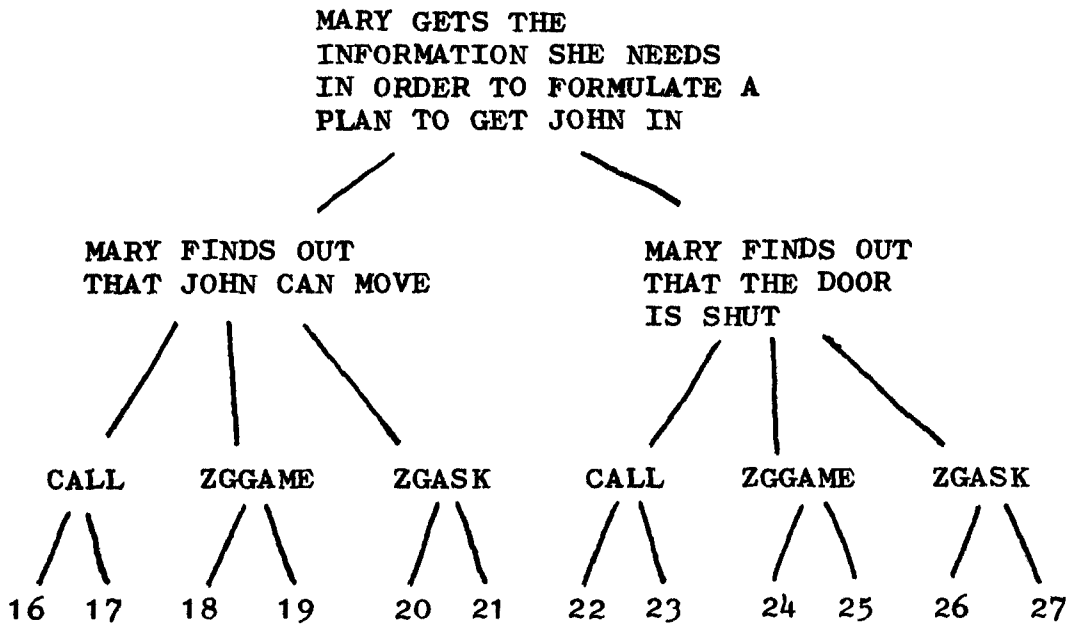
good deal more about John's mind than she does. She needs to know what he believes about the positions of objects as well as whether he can see them, and she needs to know his plans and the state of his world model in order to work out whether his question was asked for a relevant reason.

Amending the model so that it could enforce illocutionary rules would not be too difficult. Two kinds of change would be needed. First, the model of the partner would have to be extended as indicated above, and facilities for updating it provided. (For instance, if Mary tells John that the door is open, and he believes her, she must record that he now knows this new fact.) Second, the game would have to be altered so that Black, when responding to White's first move (a question, say) would make sure that the relevant illocutionary rules were obeyed before replying. If the rules were broken, then one could either allow a new kind of move in the game, or arrange for a special complaining game to be loaded by an interruption. In human usage, this might lead to a discussion in which White tried to justify his remark, but this would be much harder to arrange. The facility for detecting a violation of the illocutionary rules would only be used in conversation with a human operator, since the robots never break them.

(f) <u>The robots cannot refer to previous sections of the dialogue</u> - since utterances are essentially the entries in games - that is, the local variables of joint procedures -

they are lost as soon as the game exits. If John asks Mary if the door is open, and she says it is, then when the game is over John will know where the door is, but will have no record of the fact that Mary just told him where it is. Not only are the individual remarks lost, but also the name of the game they were part of. It would thus be impossible to adapt the system so that it could answer questions on, or make comments on, the previous dialogue. If, for example, John asks the same question twice running, it is impossible for Mary to say "I've just told you that" in reply to the second question.

The easiest way of rectifying this fault is, of course, to keep a list of all the games played and all the entries made within them. This is scarcely, however, a natural solution to the problem. Human language-users only remember distant conversations in general terms, and their memory gets gradually more general as the events in question recede. It is as if they were building one or more tree structures to describe the sequences of utterances, and the lower nodes fell off as they became more distant. For example, utterances M16 to J27 in R2 might be described as follows:



One could imagine these various levels gradually dropping off, so that at the end of the run the robots only retained the top level, or perhaps the top two. It is a weakness of the program that it cannot describe conversation at any level beyond the game level; consequently, it could not easily be adapted so that it was able to remember and refer to previous conversation in a general and natural way.

(g) <u>The robots cannot tell whether or not a game is appropriate</u> - once a game is loaded, the robots can determine whether the utterances made <u>inside</u> it are appropriate, but they cannot make this judgement about games as a whole. Thus when ZGGAME is played, Black never decides that the suggested game is not appropriate and refuses to play it; at most, he waits for a while until his routine has reached

a certain point, and then gives the go ahead. This defect arises from the fact mentioned above: namely, the inability of the program to assign a high-level description to the conversation. Since the robots never ask why the partner should want to play a given game, they cannot judge whether or not his reasons are sufficient. Criticisms (d) and (e) are also caused, in part, by this defect. In order to detect misunderstandings, or to notice violations of illocutionary rules, it is necessary to have some representation of what games are for, and thus to be able to infer the mental states which underlie them. John and Mary are wholly unable to do this: they simply load the game as they are told to, possibly after a delay; then play it; then forget it.

3.3 Improving the Model

The seven criticisms listed in 3.2 can be distilled into two underlying faults: first, the program does not know what each game is meant to achieve; and second, it does not know why the instructions making up each game are suitable for achieving the game's purpose. If the robots knew what each game was for, they would be able to detect violations of illocutionary rules, tell whether a given game was appropriate, and assign a higher level description to the dialogue. And if they knew what each instruction achieved, and thus knew why a given set of instructions achieved a given purpose,

they could design amd modify games themselves, and do away with the need to follow the same rigid definition every time.

The task, then, is to define these kinds of understanding more precisely, and to say how they can be incorporated into a program. It is useful to begin by reconsidering the method by which our program achieves physical goals - i.e. alters the values of those variables which have been chosen to represent the material world. When a robot carries out an action, it knows what it is trying to achieve, why it is trying to achieve it, and why the action is a way of achieving it. To attain this degree of understanding, it has the following facilities:

(i) It can represent states of the world by goal patterns.

(ii) It can determine whether or not a given goal is achieved.

(iii) It can devise a plan to achieve the goal, and determine whether or not a plan it is given will achieve the goal, by using its theory of the laws of nature.

(iv) It can alter this theory if it proves faulty, and thus devise a new plan if its first plan fails.

Using these facilities, a robot can describe the purpose of a given sequence of actions on several hierarchical levels - that is, it can build a planning tree - and it can explain why a given sequence should achieve a given purpose, by referring to the relevant laws of nature. This method

seems a promising way of introducing the kind of understanding that we need, and it is reasonable to ask whether it can be applied to conversations as well as to actions in the physical world.

The first step is to formulate the goals that games and their constituent instructions are meant to achieve. We will use ZGASK and ZGPLAN as examples. The goal of ZGASK is to fill a particular gap in White's world model. If John asks whether the door is open, his implicit goal is to change his model of object positions so that the symbol after DOOR is not UNDEF. This is a mental goal, but there is no difficulty in arranging a test for whether or not it is attained. There are some complications, however. The description of the goal given above is not quite accurate, since it implies that <u>any</u> updating of the model is acceptable: John might achieve the goal by tossing a coin, and putting OPEN after DOOR if it came down heads, and SHUT if tails. One could get round this by putting a restriction on the plans which John could try in order to achieve the goal. A more satisfactory solution would face up to the problem of what constitutes adequate grounds for a belief, a problem which the present model evades: when John updates his world model, he has no way of assessing or recording the credibility of the source.

Another complication which arises when we allow mental goals is the difficulty in determining the mental

state of the partner. If John wants to find out whether he knows something, we assume he can do so directly and unequivocally (though it is not certain that human language-users always can); but if he wants to find out if Mary knows it, he has to rely on various indirect clues: her answer to questions, what she does, the plans she suggests, and so on. If he tells her something, he will have to assume she now knows it, but be prepared to revise this opinion if subsequent events contradict it. So a system which tries to achieve mental goals must have facilities for weighing contradictory evidence, since the authority of direct perception can no longer be relied on.

Choosing a pattern to represent mental goals raises no special difficulty: the pattern must be designed so that it contains the information needed (a) to test whether the goal is achieved, and (b) to find a plan to achieve it. The details would depend on the way in which the conceptual system is arranged. If doors could have several properties - position, colour, size, etc. - a pattern such as

[KNOW JOHN [POSITION DOOR]]

would be appropriate. One could imagine a PLANNER-like system which had procedures for inferring and achieving statements of the general form [KNOW ?X ?Y], and which acted differently if the value of ?X was the robot itself rather than something or someone else.

Applying these methods to ZGPLAN also raises interesting problems. The goal of ZGPLAN is to add a plan to both trees: John's planning tree, and Mary's. The problems of checking are similar to those for ZGASK: John will assume that if Mary agrees to his plan she will have added the plan to her tree, but if she carries out an unexpected action in the future he will revise this opinion. An additional problem is the relationship of the goal of the game to the goals on the planning tree. The higher aim of ZGPLAN is to achieve the current goal on the planning tree, and such an aim can scarcely be attached to the planning tree itself! Thus we need two trees in each mind: the planning tree, which operates on the world, and a meta-tree which acts on the planning tree and world model. The meta-tree will perform the computations now carried out by routines and games, but will perform them more flexibly since it contains a representation of what is being done at each point. In other words, the knowledge <u>implicit</u> in routines and games will have been made <u>explicit</u>.

Relating the two trees is in fact more complicated than this suggests, since just as physical goals can be furthered by achieving mental goals, so mental goals can be furthered by achieving physical ones. When we explore a place, or carry out a scientific experiment, we are concerned with improving our knowledge, the physical actions being

of subsidiary interest. So a simple model of tree1 operating on the world and tree2 operating on tree1 will not do: we will want to have some physical goals on tree2, and then of course we will need an even higher tree to develop these goals.

To amplify these problems further would take us too far afield; instead, we will restate the general problem. The overall aim is to construct a model which can use language purposefully, and we have argued that the model described in this paper is deficient in that it has no representation of the purposes of each utterance and each game. Because of this defect, it cannot describe the dialogue in general terms, and its conversation is inflexible. The problem, as we see it, is how to give the program an explicit knowledge of both its own mentality and the mentality of its partner. The routines and games contain this knowledge implicitly: the goals of updating the world model, and getting a plan added to both trees, were goals in the programmer's mind, and the routines and games are the particular procedures he designed to achieve them. The next stage is to make these goals explicit in the program, so that the program can build a variety of routines and games (or equivalent structures). Instead of giving the robots particular formats for thinking and talking, we need to give them the knowledge underlying these formats, the

knowledge of how to generate possible formats. In doing so, we would push the program's lack of understanding back by one layer: its rules for generating routines and games would now represent the level at which it became rigid. A further program might specify rules for generating and modifying these rules, and so on. There comes a point, of course, when we have to decide to stop; there is never a rock-solid foundation, but there might be a point at which a rigid procedure almost always works. What is certain is that an adequate account of a human language-user's knowledge must go several layers deeper than the model described in this report.

3.4 Learning

"Learning" usually refers to a relatively permanent change in the mental state of an organism (or robot) due to experience, and there are thus two theoretical problems associated with it: first, describing the exact change which occurs; and second, explaining how it occurs, by specifying the intermediate processes. In this section we begin with the first problem, and consider what the programmer would need to do in order to add a new game to the robot's repertoire. Then we turn to the more difficult problem of designing a system which could pick up new games in the course of an ordinary conversation, just as a

child picks up the grammar and conversational customs of its culture.

The model described in this report has games for asking and telling, but not for commanding: thus it can produce interrogatives and declaratives, but not imperatives. Suppose that we wanted to remedy this omission by adding the game JGCOMMAND to John's repertoire. (As before, we use John for the example, and similar remarks apply to Mary). What alterations in the program will be needed if this "learning" is to take place?

(a) The game must be defined. Suppose it has just one utterance, the command made by White, and that Black responds by doing as she is told. The definition might be:

```
GAME JGCOMMAND;
1. * W JGORDER [JDACT];
2. ↑ B JGOBEY;
END;
```

(b) The functions JGORDER and JGOBEY must be defined. This is not a trivial task: JGOBEY, for instance, must take the entry in 1 ([PUSH], perhaps) and use it to build an instruction which carries out the action. JGORDER will be easier, since it only has to find the first move, already specified in JEMOVE 1 (this is the normal practice - see 2.6), and to enter it.

(c) The decoding function JDACT must be defined. There are three acts - MOVE, PUSH and SLIDE - so JDACT has to search through the English expression, and return one of these actions, or UNDEF if it fails to find any.

(d) The encoding function JWORDER must be defined; this translates an action into an English command. It might convert [PUSH] and [SLIDE] into [PUSH THE DOOR] and [SLIDE THE BOLT], leaving [MOVE] as it is.

(e) Game JGGAME must be altered so that it will arrange to play JGCOMMAND if this game is suggested. For instance, the function which decodes the first move of JGGAME must be changed so that it accepts "I want to make a command" (say) as an appropriate remark, and translates it into [JGCOMMAND].

(f) If John is to be able to play the game as White, as well as Black, his routines must be altered so that he calls the game in appropriate circumstances (e.g. when he is attempting the goal himself, and cannot perform one of the actions needed). In this case, considerable rewriting would be required if the game were to be used sensibly.

This list shows that the addition of an extra game - even a simple one like JGCOMMAND - is a major task, and not one which a program would be expected to perform. Apart from the multiplicity of the changes required, there are two difficulties: the changes have to be made at inaccessible places, and the amount of information needed to make each

change is enormous. The first of these difficulties might be circumvented by writing the functions in a special language, designed for ease of editing, but there is no way round the second. The task of knowing exactly where to edit, and exactly what instructions to write, is overwhelming: there are simply too many possibilities.

This is no new problem: Chomsky and others have drawn attention to it as one of the central problems of language. How is it that a child, in the space of five years, develops a command of its native language on the basis of evidence which seems hopelessly inadequate? According to Chomsky, "it is only by assuming that the child is born with a knowledge of the highly restrictive principles of universal grammar, and the predisposition to make use of them in analysing the utterances he hears about him, that we can make any sense of the process of language-learning" (Lyons 1970, p.106). It follows that if we want programs to use conversational clues in order to write (or to edit) their own procedures, we must restrict the range of possible alterations in the most severe manner.

In our view, the natural way of restricting hypotheses is to write the procedures in a special-purpose, highly restricted programming language. Games, in fact, are quite good from this point of view, except in one respect: the functions mentioned in their definitions (e.g. JGORDER, JGOBEY

in the game defined above) are written in POP-2, and POP-2 is a rich language, not a specially restricted one. We need to distill these functions into a few high-level primitives which can be assumed to be defined innately. Thus the program's "innate endowment" is to be a capacity to interpret, and form instructions in, a number of special-purpose languages, each language corresponding to a different aspect (i.e. level) of the language-using process. If a program like this could be written, what would the primitives be in the language for games (i.e. in the language for the procedures which conducted conversation)? The main ones would be functions to update the memory, access it, make an entry, send an entry to the underlying routine, and read a previous entry. One would need to design the primitives and the interpreter so that each instruction in the game could be completely defined by using just a few primitives.

Integrating routines and games remains a problem. One way round it is to assume that the program knows "innately" which games can exist, and that these games are mentioned in routines from the outset. What the program has to learn is the particular form of each game used by its trainer. Such a program would need to be able to handle the situation of a game being wanted by a routine, but not yet defined. A system such as that suggested in 3.3 could handle this better, as the game would be thought of as just one possible method of achieving a mental goal.

There are, of course, numerous other difficulties: the program, if intended as a model of what the child does, would have to learn all the aspects of language-use simultaneously: grammar, the conceptual system, the conversational forms, perhaps even the planning processes. It would have to segment utterances, and sequences of utterances, into appropriate units; in other words, to work out where games began and ended. When it was given feedback - told it was wrong, say - it would have to decide where the mistake occurred: whether its grammar needed changing, or its game definitions, or whatever. Our model is not well designed with respect to these tasks: its knowledge is not represented in a form which could be readily learned or modified. It is likely that a more advanced model, one which had a deeper understanding of its own mentality (see 3.3), would also be much easier to convert into a learning model, since it would represent the knowledge in a clean, explicit way instead of burying it in complicated POP-2 functions. There is not much point in attempting detailed learning models while our models of the "adult" language-user are so primitive.

3.5 Parallel Processing

At present, the model is set up so that the robots only pass control to each other when they cannot carry on with their private thoughts, the reason usually being that they

want to give the partner a chance to say something. Thus a robot can keep control for long periods, and during these periods it cannot be interrupted. It recently occurred to me that a much better arrangement was possible: the robots could swap control inbetween every instruction of the currently running game or routine. The program could easily be adapted to work this way, and the result would be a much nearer approximation to a truly parallel situation.

To adapt the program in this way, one would have to change the executive (see 2.7) so that whenever an instruction was completed, control returned to the chairman, instead of looping back in order to tackle the next instruction. We will consider the first few utterances of R2 in order to show how such a system would work. This example is fully described (for the priginal model) in section 2.6.2. To describe the performance of the adapted model, we will give a step by step account of the routine instructions carried out, with occasional notes. Each step is numbered, the robot concerned is mentioned (J or M), and the instruction and entry (if any) are given. Remember that the entry for JRBASIC 4 has been preset to [FAILED] to stop John attempting the goal on his own.

1. J JRBASIC 1 (John prepares his world model)
2. M MRBASIC 1
3. J JRBASIC 2 [JOHN IN] (John enters his main goal)

4. M MRBASIC 2 [NONE]

5. J JRBASIC 3 (John goes to 4 to attempt goal alone)

6. M MRBASIC 3 (Mary has no goal, so goes to 9 to halt)

7. J JRBASIC 5 (The entry in 4 was already [FAILED])

8. M MRBASIC 9

9. J JRBASIC 6 (John loads JGGAME to suggest JGGOAL)

John now calls Mary, the first utterance. Mary loads MGGAME and replies "Yes", without a game or routine instruction being involved.

10. M (Loads MGGAME)

11. J JGGAME 1 [JGGOAL] (John wants to suggest a goal)

12. M MGGAME 2 [YES] (Mary loads MGGOAL)

Mary has entered [MGGOAL] at MGGAME 1, replied, and loaded the new game.

13. J JGGAME 3 (John loads JGGOAL)

14. M (returns control as first move in MGGOAL is John's)

15. J JGGOAL 1 [JOHN IN]

16. M MGGOAL 2 [YES] (Mary moves to MRBASIC 7)

17. J JGGOAL 3 (John returns [YES] to JRBASIC 6)

18. M MRBASIC 7 (Mary gets ready to attempt goal jointly)

19. J JRBASIC 7

20. M MRBASIC 8 (Loads MRACHIEVE)

21. J JRBASIC 8

22. M MRACHIEVE 1 [JOHN IN] (enters current goal)

And so on. Although this method is a closer approximation

to parallel processing, it is not necessarily more natural, since the robots tend to be thinking too closely in concert, only separated by the odd instruction. One of the strengths of the program is that it is not fussy about which time-sharing system is used: each robot saves its place all the time, and could be interrupted at any point. If the program is conversing with a human operator, the operator could regain control at any time, even if the program was deep inside a function (there is a facility for doing this by depressing a particular teletype key); he could then change the value of WMESSAGE in order to make an utterance, and recall the program. This would not put the program out in any way: if it expected the utterance, or the utterance was a call, it would deal with it; if not, it would carry on with its current routine. Thus the model is effectively a parallel one; in other words, there are no theoretical problems involved in making it work in parallel. The only situation it could not handle would be the situation in which both parties spoke at once: human speakers usually react by starting again, the decision as to who starts being arbitrary or a reflection of personality differences. At present the model has no facility for restarting in this way, but it is not a major problem.

APPENDICES

Appendix I: Output

This appendix gives the two output examples mentioned in 2.6.2, and used there to illustrate how games work. They are called R1 and R2, R1 is a copy of the example given in 1.4, without annotations; and R2 is another copy in which a full print-out of the progress of all the routines and games is given as well as the utterances.

*** [R1] ***

(STATE OF WORLD IS NOW [JOHN OUT MARY IN BOLT UP DOOR SHUT])

1 JOHN: MARY.

2 MARY: YES.

3 JOHN: I WANT TO SUGGEST A GOAL.

4 MARY: GO AHEAD.

5 JOHN: WILL YOU HELP ME GET IN.

6 MARY: BY ALL MEANS.

7 JOHN: SHALL WE MAKE A PLAN.

8 MARY: JOHN.

9 JOHN: YES.

10 MARY: MAY I ASK YOU SOMETHING.

11 JOHN: GO AHEAD.

12 MARY: ARE YOU IN.

13 JOHN: NO.

14 MARY: SHALL WE MAKE A PLAN.

15 JOHN: OK.

16 MARY: JOHN.

17 JOHN: YES.

18 MARY: MAY I ASK YOU SOMETHING.

19 JOHN: GO AHEAD.

20 MARY: CAN YOU MOVE.

21 JOHN: YES.

22 MARY: JOHN.

23 JOHN: YES.

24 MARY: MAY I ASK YOU SOMETHING.

25 JOHN: GO AHEAD.

26 MARY: IS THE DOOR OPEN.

27 JOHN: NO.

28 MARY: I SUGGEST THAT WE GET THE DOOR OPEN AND THEN YOU MOVE.

29 JOHN: MARY.

30 MARY: YES.

31 JOHN: I WANT TO EXPLAIN SOMETHING.

32 MARY: GO AHEAD.

33 JOHN: IF YOU MOVE, NOTHING HAPPENS.

34 MARY: I DISAGREE. IF YOU MOVE WHEN THE DOOR IS OPEN, YOU CHANGE POSITION.

35 JOHN: I SEE.

36 MARY: I SUGGEST THAT WE GET THE DOOR OPEN AND THEN YOU MOVE.

37 JOHN: ALL RIGHT.

38 MARY: SHALL WE MAKE A PLAN.

39 JOHN: OK.

40 MARY: I SUGGEST THAT I PUSH THE DOOR.

41 JOHN: ALL RIGHT.

(STATE OF WORLD IS NOW [DOOR OPEN JOHN OUT MARY IN BOLT UP])

42 MARY: I WANT TO TELL YOU SOMETHING.

43 JOHN: GO AHEAD.

44 MARY: I HAVE PUSHED THE DOOR.

45 JOHN: I SEE.

46 MARY: LETS ASSESS THE RESULT OF MY ACTION.

47 JOHN: OK.

48 MARY: NOTHING HAS HAPPENED.

49 JOHN: MARY.

50 MARY: YES.

51 JOHN: I WANT TO TELL YOU SOMETHING.

52 MARY: GO AHEAD.

53 JOHN: THE DOOR IS OPEN.

54 MARY: I SEE. THE DOOR HAS CHANGED POSITION.

55 JOHN: YES.

56 MARY: THE DOOR IS NOW OPEN.

57 JOHN: RIGHT.

(STATE OF WORLD IS NOW [JOHN IN MARY IN BOLT UP DOOR OPEN])

57 JOHN: I WANT TO TELL YOU SOMETHING.

58 MARY: GO AHEAD.

59 JOHN: I HAVE MOVED.

60 MARY: I SEE.

61 JOHN: LETS ASSESS THE RESULT OF MY ACTION.

62 MARY: OK.

63 JOHN: I HAVE CHANGED POSITION.

64 MARY: JOHN.

65 JOHN: YES.

66 MARY: I WANT TO TELL YOU SOMETHING.

67 JOHN: GO AHEAD.

68 MARY: YOU ARE OUT.

69 JOHN: I DISAGREE. I HAVE CHANGED POSITION.

70 MARY: YES.

71 JOHN: I AM NOW IN.

72 MARY: RIGHT.

```
*** [R2] ***


 JRBASIC LOADED BY JOHN
 MRBASIC LOADED BY MARY

(STATE OF WORLD IS NOW [JOHN OUT MARY IN BOLT UP DOOR SHUT])

 1 CALLED BY JOHN
 2 CALLED BY JOHN
 2 [JOHN IN] ENTERED BY JOHN
 3 CALLED BY JOHN
 5 CALLED BY JOHN
 6 CALLED BY JOHN
   JGGAME LOADED BY JOHN

** 1 JOHN: MARY.

   MGGAME LOADED BY MARY

** 2 MARY: YES.

   1 CALLED BY JOHN
   1 [JGGOAL] ENTERED BY JOHN

** 3 JOHN: I WANT TO SUGGEST A GOAL.

   1 [MGGOAL] ENTERED BY MARY
   2 CALLED BY MARY
 1 CALLED BY MARY
 2 CALLED BY MARY
 2 [NONE] ENTERED BY MARY
   2 CALLED BY MARY
   2 [YES] ENTERED BY MARY
```

MGGAME ENDED BY MARY

MGGOAL LOADED BY MARY

** 4 MARY: GO AHEAD.

2 [YES] ENTERED BY JOHN

3 CALLED BY JOHN

JGGAME ENDED BY JOHN

JGGOAL LOADED BY JOHN

1 CALLED BY JOHN

1 [JOHN IN] ENTERED BY JOHN

** 5 JOHN: WILL YOU HELP ME GET IN.

1 [JOHN IN] ENTERED BY MARY

2 CALLED BY MARY

2 [YES] ENTERED BY MARY

** 6 MARY: BY ALL MEANS.

2 [YES] ENTERED BY JOHN

3 CALLED BY JOHN

JGGOAL ENDED BY JOHN

6 [YES] ENTERED BY JOHN

7 CALLED BY JOHN

8 CALLED BY JOHN

JRACHIEV LOADED BY JOHN

1 CALLED BY JOHN

1 [JOHN IN] ENTERED BY JOHN

2 CALLED BY JOHN

2 [BOTH] ENTERED BY JOHN

3 CALLED BY JOHN

3 [SITN] ENTERED BY JOHN

```
   4 CALLED BY JOHN
   4 [NOTYET] ENTERED BY JOHN
   5 CALLED BY JOHN
     JGGAME LOADED BY JOHN
     1 CALLED BY JOHN
     1 [JGPLAN] ENTERED BY JOHN

** 7 JOHN: SHALL WE MAKE A PLAN.

   MGGAME LOADED BY MARY
   1 [MGPLAN] ENTERED BY MARY
   2 CALLED BY MARY
 7 CALLED BY MARY
 8 CALLED BY MARY
   MRACHIEV LOADED BY MARY
   1 CALLED BY MARY
   1 [JOHN IN] ENTERED BY MARY
   2 CALLED BY MARY
   2 [BOTH] ENTERED BY MARY
   3 CALLED BY MARY
   3 [SITN] ENTERED BY MARY
   4 CALLED BY MARY
     MGGAME LOADED BY MARY

** 8 MARY: JOHN.

       JGGAME LOADED BY JOHN

** 9 JOHN: YES.

     1 CALLED BY MARY
     1 [MGASK] ENTERED BY MARY
```

```
** 10 MARY: MAY I ASK YOU SOMETHING.

     1 [JGASK] ENTERED BY JOHN
     2 CALLED BY JOHN
     2 [YES] ENTERED BY JOHN
     JGGAME ENDED BY JOHN
     JGASK LOADED BY JOHN

** 11 JOHN: GO AHEAD.

     2 [YES] ENTERED BY MARY
     3 CALLED BY MARY
     MGGAME ENDED BY MARY
     MGASK LOADED BY MARY
     1 CALLED BY MARY
     1 [IS JOHN IN] ENTERED BY MARY

** 12 MARY: ARE YOU IN.

     1 [IS JOHN IN] ENTERED BY JOHN
     2 CALLED BY JOHN
     2 [NO] ENTERED BY JOHN
     JGASK ENDED BY JOHN

** 13 JOHN: NO.

     2 [NO] ENTERED BY MARY
     3 CALLED BY MARY
     MGASK ENDED BY MARY
   4 CALLED BY MARY
   4 [NOTYET] ENTERED BY MARY
   5 CALLED BY MARY
     MGGAME LOADED BY MARY
     1 CALLED BY MARY
```

```
      1 [MGPLAN] ENTERED BY MARY

** 14 MARY: SHALL WE MAKE A PLAN.

      JGGAME LOADED BY JOHN
      1 [JGPLAN] ENTERED BY JOHN
      2 CALLED BY JOHN
      2 [YES] ENTERED BY JOHN
      JGGAME ENDED BY JOHN
      JGPLAN LOADED BY JOHN

** 15 JOHN: OK.

      2 [YES] ENTERED BY MARY
      3 CALLED BY MARY
      MGGAME ENDED BY MARY
      MGPLAN LOADED BY MARY
      1 CALLED BY MARY
        MRPLAN LOADED BY MARY
        1 CALLED BY MARY
        1 [JOHN IN] ENTERED BY MARY
        2 CALLED BY MARY
        2 [EVT [ROBOT MOVE] SIT [DOOR OPEN] RES [ROBOT]] ENTERED BY MARY
        3 CALLED BY MARY
          MGGAME LOADED BY MARY

** 16 MARY: JOHN.

       JGGAME LOADED BY JOHN

** 17 JOHN: YES.

         1 CALLED BY MARY
         1 [MGASK] ENTERED BY MARY
```

** 18 MARY: MAY I ASK YOU SOMETHING.

1 [JGASK] ENTERED BY JOHN

2 CALLED BY JOHN

2 [YES] ENTERED BY JOHN

JGGAME ENDED BY JOHN

JGASK LOADED BY JOHN

** 19 JOHN: GO AHEAD.

2 [YES] ENTERED BY MARY

3 CALLED BY MARY

MGGAME ENDED BY MARY

MGASK LOADED BY MARY

1 CALLED BY MARY

1 [CAN JOHN MOVE] ENTERED BY MARY

** 20 MARY: CAN YOU MOVE.

1 [CAN JOHN MOVE] ENTERED BY JOHN

2 CALLED BY JOHN

2 [YES] ENTERED BY JOHN

JGASK ENDED BY JOHN

** 21 JOHN: YES.

2 [YES] ENTERED BY MARY

3 CALLED BY MARY

MGASK ENDED BY MARY

3 CALLED BY MARY

3 [EVT [JOHN MOVE] SIT [DOOR OPEN] RES [JOHN]] ENTERED BY MARY

4 CALLED BY MARY

MGGAME LOADED BY MARY

** 22 MARY: JOHN.

JGGAME LOADED BY JOHN

** 23 JOHN: YES.

1 CALLED BY MARY

1 [MGASK] ENTERED BY MARY

** 24 MARY: MAY I ASK YOU SOMETHING.

1 [JGASK] ENTERED BY JOHN

2 CALLED BY JOHN

2 [YES] ENTERED BY JOHN

JGGAME ENDED BY JOHN

JGASK LOADED BY JOHN

** 25 JOHN: GO AHEAD.

2 [YES] ENTERED BY MARY

3 CALLED BY MARY

MGGAME ENDED BY MARY

MGASK LOADED BY MARY

1 CALLED BY MARY

1 [IS DOOR OPEN] ENTERED BY MARY

** 26 MARY: IS THE DOOR OPEN.

1 [IS DOOR OPEN] ENTERED BY JOHN

2 CALLED BY JOHN

2 [NO] ENTERED BY JOHN

JGASK ENDED BY JOHN

** 27 JOHN: NO.

```
        2 [NO] ENTERED BY MARY
        3 CALLED BY MARY
        MGASK ENDED BY MARY
      4 CALLED BY MARY
      4 [NOTYET] ENTERED BY MARY
      5 CALLED BY MARY
      MRPLAN ENDED BY MARY
    1 [BOTH [DOOR OPEN] JOHN [JOHN MOVE]] ENTERED BY MARY

** 28 MARY: I SUGGEST THAT WE GET THE DOOR OPEN AND THEN YOU MOVE.

    1 [BOTH [DOOR OPEN] JOHN [JOHN MOVE]] ENTERED BY JOHN
    2 CALLED BY JOHN
      JGGAME LOADED BY JOHN

** 29 JOHN: MARY.

      MGGAME LOADED BY MARY

** 30 MARY: YES.

      1 CALLED BY JOHN
      1 [JGRULE] ENTERED BY JOHN

** 31 JOHN: I WANT TO EXPLAIN SOMETHING.

      1 [MGRULE] ENTERED BY MARY
      2 CALLED BY MARY
      2 [YES] ENTERED BY MARY
      MGGAME ENDED BY MARY
      MGRULE LOADED BY MARY

** 32 MARY: GO AHEAD.

      2 [YES] ENTERED BY JOHN
```

```
        3 CALLED BY JOHN
        JGGAME ENDED BY JOHN
        JGRULE LOADED BY JOHN
        1 CALLED BY JOHN
        1 [EVT [ROBOT MOVE] SIT [ANY] RES [NOTHING]] ENTERED BY JOHN

** 33 JOHN: IF YOU MOVE, NOTHING HAPPENS.

        1 [EVT [ROBOT MOVE] SIT [ANY] RES [NOTHING]] ENTERED BY MARY
        2 CALLED BY MARY
        2 [NO] ENTERED BY MARY

** 34 MARY: I DISAGREE.

        2 [NO] ENTERED BY JOHN
        3 CALLED BY JOHN

** JOHN SWAPS

        4 CALLED BY MARY
        4 [EVT [ROBOT MOVE] SIT [DOOR OPEN] RES [ROBOT]] ENTERED BY MARY

** 34 MARY: IF YOU MOVE WHEN THE DOOR IS OPEN, YOU CHANGE POSITION.

        4 [EVT [ROBOT MOVE] SIT [DOOR OPEN] RES [ROBOT]] ENTERED BY JOHN
        5 CALLED BY JOHN
        5 [UNDEF] ENTERED BY JOHN
        JGRULE ENDED BY JOHN

** 35 JOHN: I SEE.

        5 [UNDEF] ENTERED BY MARY
        6 CALLED BY MARY
        MGRULE ENDED BY MARY
      1 CALLED BY MARY
```

```
      MRPLAN LOADED BY MARY
      1 CALLED BY MARY
      1 [JOHN IN] ENTERED BY MARY
      2 CALLED BY MARY
      2 [EVT [ROBOT MOVE] SIT [DOOR OPEN] RES [ROBOT]] ENTERED BY MARY
      3 CALLED BY MARY
      3 [EVT [JOHN MOVE] SIT [DOOR OPEN] RES [JOHN]] ENTERED BY MARY
      4 CALLED BY MARY
      4 [NOTYET] ENTERED BY MARY
      5 CALLED BY MARY
      MRPLAN ENDED BY MARY
    1 [BOTH [DOOR OPEN] JOHN [JOHN MOVE]] ENTERED BY MARY

** 36 MARY: I SUGGEST THAT WE GET THE DOOR OPEN AND THEN YOU MOVE.

    1 [BOTH [DOOR OPEN] JOHN [JOHN MOVE]] ENTERED BY JOHN
    2 CALLED BY JOHN
    2 [YES] ENTERED BY JOHN
    JGPLAN ENDED BY JOHN
  5 [BOTH [DOOR OPEN] JOHN [JOHN MOVE]] ENTERED BY JOHN

** 37 JOHN: ALL RIGHT.

    2 [YES] ENTERED BY MARY
    3 CALLED BY MARY
    MGPLAN ENDED BY MARY
  5 [BOTH [DOOR OPEN] JOHN [JOHN MOVE]] ENTERED BY MARY
  6 CALLED BY MARY
  MRACHIEV ENDED BY MARY
  MRACHIEV LOADED BY MARY
  1 CALLED BY MARY
  1 [DOOR OPEN] ENTERED BY MARY
```

```
    2 CALLED BY MARY
    2 [BOTH] ENTERED BY MARY
    3 CALLED BY MARY
    3 [SITN] ENTERED BY MARY
    4 CALLED BY MARY
    4 [NOTYET] ENTERED BY MARY
    5 CALLED BY MARY
      MGGAME LOADED BY MARY
      1 CALLED BY MARY
      1 [MGPLAN] ENTERED BY MARY

** 38 MARY: SHALL WE MAKE A PLAN.

      JGGAME LOADED BY JOHN
      1 [JGPLAN] ENTERED BY JOHN
      2 CALLED BY JOHN
   6 CALLED BY JOHN
   JRACHIEV ENDED BY JOHN
   JRACHIEV LOADED BY JOHN
   1 CALLED BY JOHN
   1 [DOOR OPEN] ENTERED BY JOHN
   2 CALLED BY JOHN
   2 [BOTH] ENTERED BY JOHN
   3 CALLED BY JOHN
   3 [SITN] ENTERED BY JOHN
   4 CALLED BY JOHN
   4 [NOTYET] ENTERED BY JOHN
     2 CALLED BY JOHN
     2 [YES] ENTERED BY JOHN
     JGGAME ENDED BY JOHN
     JGPLAN LOADED BY JOHN
```

** 39 JOHN: OK.

2 [YES] ENTERED BY MARY

3 CALLED BY MARY

MGGAME ENDED BY MARY

MGPLAN LOADED BY MARY

1 CALLED BY MARY

MRPLAN LOADED BY MARY

1 CALLED BY MARY

1 [DOOR OPEN] ENTERED BY MARY

2 CALLED BY MARY

2 [EVT [ROBOT PUSH] SIT [ANY] RES [NOTHING]] ENTERED BY MARY

3 CALLED BY MARY

3 [EVT [MARY PUSH] SIT [ANY] RES [NOTHING]] ENTERED BY MARY

4 CALLED BY MARY

4 [ACHIEVED] ENTERED BY MARY

5 CALLED BY MARY

MRPLAN ENDED BY MARY

1 [MARY [MARY PUSH]] ENTERED BY MARY

** 40 MARY: I SUGGEST THAT I PUSH THE DOOR.

1 [MARY [MARY PUSH]] ENTERED BY JOHN

2 CALLED BY JOHN

2 [YES] ENTERED BY JOHN

JGPLAN ENDED BY JOHN

5 [MARY [MARY PUSH]] ENTERED BY JOHN

** 41 JOHN: ALL RIGHT.

2 [YES] ENTERED BY MARY

3 CALLED BY MARY

```
   MGPLAN ENDED BY MARY
 5 [MARY [MARY PUSH]] ENTERED BY MARY
 6 CALLED BY MARY
 MRACHIEV ENDED BY MARY
 MRACHIEV LOADED BY MARY
 1 CALLED BY MARY
 1 [MARY PUSH] ENTERED BY MARY
 2 CALLED BY MARY
 2 [MARY] ENTERED BY MARY
 3 CALLED BY MARY
 3 [EVENT] ENTERED BY MARY
 7 CALLED BY MARY
 7 [DOOR SHUT BOLT UNDEF JOHN OUT MARY IN] ENTERED BY MARY
 8 CALLED BY MARY

(STATE OF WORLD IS NOW [DOOR OPEN JOHN OUT MARY IN BOLT UP])

 8 [DONE] ENTERED BY MARY
   MGGAME LOADED BY MARY
   1 CALLED BY MARY
   1 [MGTELL] ENTERED BY MARY

** 42 MARY: I WANT TO TELL YOU SOMETHING.

   JGGAME LOADED BY JOHN
   1 [JGTELL] ENTERED BY JOHN
   2 CALLED BY JOHN
   2 [YES] ENTERED BY JOHN
   JGGAME ENDED BY JOHN
   JGTELL LOADED BY JOHN

** 43 JOHN: GO AHEAD.
```

```
      2 [YES] ENTERED BY MARY
      3 CALLED BY MARY
      MGGAME ENDED BY MARY
      MGTELL LOADED BY MARY
      1 CALLED BY MARY
      1 [MARY PUSH] ENTERED BY MARY

** 44 MARY: I HAVE PUSHED THE DOOR.

      1 [MARY PUSH] ENTERED BY JOHN
      2 CALLED BY JOHN
    6 CALLED BY JOHN
    JRACHIEV ENDED BY JOHN
    JRACHIEV LOADED BY JOHN
    1 CALLED BY JOHN
    1 [MARY PUSH] ENTERED BY JOHN
    2 CALLED BY JOHN
    2 [MARY] ENTERED BY JOHN
    3 CALLED BY JOHN
    3 [EVENT] ENTERED BY JOHN
    7 CALLED BY JOHN
    7 [MARY IN DOOR SHUT BOLT UP JOHN OUT] ENTERED BY JOHN
      2 CALLED BY JOHN
      2 [UNDEF] ENTERED BY JOHN
      JGTELL ENDED BY JOHN
    8 [DONE] ENTERED BY JOHN

** 45 JOHN: I SEE.

      2 [UNDEF] ENTERED BY MARY
      3 CALLED BY MARY
      MGTELL ENDED BY MARY
```

9 CALLED BY MARY

MGGAME LOADED BY MARY

1 CALLED BY MARY

1 [MGASSESS] ENTERED BY MARY

** 46 MARY: LETS ASSESS THE RESULT OF MY ACTION.

JGGAME LOADED BY JOHN

1 [JGASSESS] ENTERED BY JOHN

2 CALLED BY JOHN

2 [YES] ENTERED BY JOHN

JGGAME ENDED BY JOHN

JGASSESS LOADED BY JOHN

** 47 JOHN: OK.

2 [YES] ENTERED BY MARY

3 CALLED BY MARY

MGGAME ENDED BY MARY

MGASSESS LOADED BY MARY

1 CALLED BY MARY

1 [NOTHING] ENTERED BY MARY

** 48 MARY: NOTHING HAS HAPPENED.

1 [NOTHING] ENTERED BY JOHN

2 CALLED BY JOHN

JGGAME LOADED BY JOHN

** 49 JOHN: MARY.

MGGAME LOADED BY MARY

** 50 MARY: YES.

```
        1 CALLED BY JOHN
        1 [JGTELL] ENTERED BY JOHN

** 51 JOHN: I WANT TO TELL YOU SOMETHING.

        1 [MGTELL] ENTERED BY MARY
        2 CALLED BY MARY
        2 [YES] ENTERED BY MARY
        MGGAME ENDED BY MARY
        MGTELL LOADED BY MARY

** 52 MARY: GO AHEAD.

        2 [YES] ENTERED BY JOHN
        3 CALLED BY JOHN
        JGGAME ENDED BY JOHN
        JGTELL LOADED BY JOHN
        1 CALLED BY JOHN
        1 [ 1 IS DOOR OPEN] ENTERED BY JOHN

** 53 JOHN: THE DOOR IS OPEN.

        1 [ 1 IS DOOR OPEN] ENTERED BY MARY
        2 CALLED BY MARY
        2 [UNDEF] ENTERED BY MARY
        MGTELL ENDED BY MARY

** 54 MARY: I SEE.

        2 [UNDEF] ENTERED BY JOHN
        3 CALLED BY JOHN
        JGTELL ENDED BY JOHN

** JOHN SWAPS
```

1 CALLED BY MARY

1 [DOOR] ENTERED BY MARY

** 54 MARY: THE DOOR HAS CHANGED POSITION.

1 [DOOR] ENTERED BY JOHN

2 CALLED BY JOHN

2 [YES] ENTERED BY JOHN

** 55 JOHN: YES.

2 [YES] ENTERED BY MARY

3 CALLED BY MARY

3 [1 IS DOOR OPEN] ENTERED BY MARY

** 56 MARY: THE DOOR IS NOW OPEN.

3 [1 IS DOOR OPEN] ENTERED BY JOHN

4 CALLED BY JOHN

4 [YES] ENTERED BY JOHN

JGASSESS ENDED BY JOHN

9 [DOOR] ENTERED BY JOHN

10 [ACHIEVED] ENTERED BY JOHN

** 57 JOHN: RIGHT.

4 [YES] ENTERED BY MARY

5 CALLED BY MARY

MGASSESS ENDED BY MARY

9 [DOOR] ENTERED BY MARY

10 [ACHIEVED] ENTERED BY MARY

11 CALLED BY MARY

11 [LEARNED] ENTERED BY MARY

12 CALLED BY MARY

```
   MRACHIEV ENDED BY MARY
   MRACHIEV LOADED BY MARY
   1 CALLED BY MARY
   1 [JOHN MOVE] ENTERED BY MARY
   2 CALLED BY MARY
   2 [JOHN] ENTERED BY MARY
   3 CALLED BY MARY
   3 [EVENT] ENTERED BY MARY
   7 CALLED BY MARY
   7 [MARY IN DOOR OPEN BOLT UNDEF JOHN OUT] ENTERED BY MARY
   8 CALLED BY MARY

** MARY SWAPS

   11 CALLED BY JOHN
   11 [LEARNED] ENTERED BY JOHN
   12 CALLED BY JOHN
   JRACHIEV ENDED BY JOHN
   JRACHIEV LOADED BY JOHN
   1 CALLED BY JOHN
   1 [JOHN MOVE] ENTERED BY JOHN
   2 CALLED BY JOHN
   2 [JOHN] ENTERED BY JOHN
   3 CALLED BY JOHN
   3 [EVENT] ENTERED BY JOHN
   7 CALLED BY JOHN
   7 [MARY IN DOOR OPEN BOLT UP JOHN OUT] ENTERED BY JOHN
   8 CALLED BY JOHN

(STATE OF WORLD IS NOW [JOHN IN MARY IN BOLT UP DOOR OPEN])

   8 [DONE] ENTERED BY JOHN
```

```
      JGGAME LOADED BY JOHN
      1 CALLED BY JOHN
      1 [JGTELL] ENTERED BY JOHN

** 57 JOHN: I WANT TO TELL YOU SOMETHING.

      MGGAME LOADED BY MARY
      1 [MGTELL] ENTERED BY MARY
      2 CALLED BY MARY
      2 [YES] ENTERED BY MARY
      MGGAME ENDED BY MARY
      MGTELL LOADED BY MARY

** 58 MARY: GO AHEAD.

      2 [YES] ENTERED BY JOHN
      3 CALLED BY JOHN
      JGGAME ENDED BY JOHN
      JGTELL LOADED BY JOHN
      1 CALLED BY JOHN
      1 [JOHN MOVE] ENTERED BY JOHN

** 59 JOHN: I HAVE MOVED.

      1 [JOHN MOVE] ENTERED BY MARY
      2 CALLED BY MARY
      2 [UNDEF] ENTERED BY MARY
      MGTELL ENDED BY MARY
    8 [DONE] ENTERED BY MARY

** 60 MARY: I SEE.

      2 [UNDEF] ENTERED BY JOHN
      3 CALLED BY JOHN
```

JGTELL ENDED BY JOHN

9 CALLED BY JOHN

JGGAME LOADED BY JOHN

1 CALLED BY JOHN

1 [JGASSESS] ENTERED BY JOHN

** 61 JOHN: LETS ASSESS THE RESULT OF MY ACTION.

MGGAME LOADED BY MARY

1 [MGASSESS] ENTERED BY MARY

2 CALLED BY MARY

2 [YES] ENTERED BY MARY

MGGAME ENDED BY MARY

MGASSESS LOADED BY MARY

** 62 MARY: OK.

2 [YES] ENTERED BY JOHN

3 CALLED BY JOHN

JGGAME ENDED BY JOHN

JGASSESS LOADED BY JOHN

1 CALLED BY JOHN

1 [JOHN] ENTERED BY JOHN

** 63 JOHN: I HAVE CHANGED POSITION.

1 [JOHN] ENTERED BY MARY

2 CALLED BY MARY

MGGAME LOADED BY MARY

** 64 MARY: JOHN.

JGGAME LOADED BY JOHN

** 65 JOHN: YES.

1 CALLED BY MARY

1 [MGTELL] ENTERED BY MARY

** 66 MARY: I WANT TO TELL YOU SOMETHING.

1 [JGTELL] ENTERED BY JOHN

2 CALLED BY JOHN

2 [YES] ENTERED BY JOHN

JGGAME ENDED BY JOHN

JGTELL LOADED BY JOHN

** 67 JOHN: GO AHEAD.

2 [YES] ENTERED BY MARY

3 CALLED BY MARY

MGGAME ENDED BY MARY

MGTELL LOADED BY MARY

1 CALLED BY MARY

1 [1 IS JOHN OUT] ENTERED BY MARY

** 68 MARY: YOU ARE OUT.

1 [1 IS JOHN OUT] ENTERED BY JOHN

2 CALLED BY JOHN

2 [NO] ENTERED BY JOHN

JGTELL ENDED BY JOHN

** 69 JOHN: I DISAGREE.

2 [NO] ENTERED BY MARY

3 CALLED BY MARY

MGTELL ENDED BY MARY

```
** MARY SWAPS

     1 CALLED BY JOHN
     1 [JOHN] ENTERED BY JOHN

** 69 JOHN: I HAVE CHANGED POSITION.

     1 [JOHN] ENTERED BY MARY
     2 CALLED BY MARY
     2 [YES] ENTERED BY MARY

** 70 MARY: YES.

     2 [YES] ENTERED BY JOHN
     3 CALLED BY JOHN
     3 [ 1 IS JOHN IN] ENTERED BY JOHN

** 71 JOHN: I AM NOW IN.

     3 [ 1 IS JOHN IN] ENTERED BY MARY
     4 CALLED BY MARY
     4 [YES] ENTERED BY MARY
     MGASSESS ENDED BY MARY
   9 [JOHN] ENTERED BY MARY
  10 [ACHIEVED] ENTERED BY MARY

** 72 MARY: RIGHT.

     4 [YES] ENTERED BY JOHN
     5 CALLED BY JOHN
     JGASSESS ENDED BY JOHN
   9 [JOHN] ENTERED BY JOHN
  10 [ACHIEVED] ENTERED BY JOHN
  11 CALLED BY JOHN
  11 [LEARNED] ENTERED BY JOHN
```

```
   12 CALLED BY JOHN
   JRACHIEV ENDED BY JOHN
 8 [ACHIEVED] ENTERED BY JOHN
 9 CALLED BY JOHN

** JOHN SWAPS

   11 CALLED BY MARY
   11 [LEARNED] ENTERED BY MARY
   12 CALLED BY MARY
   MRACHIEV ENDED BY MARY
 8 [ACHIEVED] ENTERED BY MARY
 9 CALLED BY MARY
```

Appendix II: Program

This appendix gives a full print-out of the program code. The program is arranged under a number of file headings, these headings being flanked by three asterisks on either side in order to distinguish them from the actual code. The program has been liberally annotated with "Comments" messages so as to make it less opaque; these messages are for human benefit only and are ignored by the computer. The files have been arranged in an order which roughly corresponds to the order of description in the text. All of the files in the program have been included; thus all the functions mentioned in the files will be defined, except for those belonging to the POP-2 system. Descriptions of these system functions are given in "Programming in POP-2" (Burstall et al, 1972).

The reader may be interested to know how the output examples in Appendix I were obtained. First, the program was compiled. Files [MACROS], [SYSTEM], [WORLD], and [PLAY] were compiled as printed, but the other files were compiled twice, substituting J for Z the first time and M for Z the second. Then the operator set the variable PRO (see file [PLAY]) to ∅, to suppress the functions which print out the progress reports for routines and games, and called function RUN1, which is defined in [PLAY]. R1 was then printed out. Then the operator set PRO to 1, and recalled RUN1, causing R2 to be printed out.

```
*** [MACROS] ***

COMMENT 'THESE MACROS ARE USED TO DEFINE CONSTANTS, GAMES
         AND ROUTINES';



MACRO CONSTANTS;
 VARS X;
C1:
 .ITEMREAD->X;
 IF X/=";" THEN [%X%]->X;
 MACRESULTS([VARS]<>X<>[; "]<>X<>[" ->]<>X<>[;]); GOTO C1
 CLOSE;
END;



MACRO GAME;
 VARS NAME X G L;
 .ITEMREAD->NAME; NIL,NIL->G->L; .ITEMREAD.ERASE;
G1:
 .ITEMREAD->X;
 IF X="B" THEN "BLACK"->X CLOSE; IF X="W" THEN "WHITE"->X CLOSE;
 IF X/="END"
 THEN IF X/=";"
      THEN IF X/="."
           THEN L<>[%X%]->L
           CLOSE;
           GOTO G1
      ELSE G<>[%L.HD,L.TL%]->G; NIL->L; GOTO G1
      CLOSE;
 ELSE MACRESULTS([%"VARS",NAME,";",G,"->",NAME%])
 CLOSE;
END;



MACRO ROUTINE; MACRESULTS([GAME]); END;
```

```
*** [SYSTEM] ***


CANCEL = ;
CANCEL /= ;
OPERATION 2 = X Y; EQUAL(X,Y) END;
OPERATION 2 /= X Y; NOT(EQUAL(X,Y)) END;



CONSTANTS JOHN MARY DICK BOTH DOOR BOLT IN OUT OPEN SHUT UP
          DOWN NAME KIND COLOUR PLACE ENTRIES MARK WHITE BLACK
          ACHIEVED FAILED NOTYET PUSH SLIDE MOVE SIT RES EVT;



COMMENT 'THE FOLLOWING FUNCTIONS ARE USED GENERALLY AS IF IN THE
         LANGUAGE. THE LETTERS USED FOR THEIR ARGUMENTS ARE MEANT
         TO INDICATE THE KIND OF VALUE THAT IS APPROPRIATE.
         X MEANS WORD, L LIST, N INDEX (USUALLY NUMERICAL),
         AND W A WORD WHICH IS A VARIABLE NAME';



COMMENT 'MEMB IS TRUE IF X IS MEMBER OF L AND FALSE IF NOT';

FUNCTION MEMB X L;
 LOOPIF L.ISLINK AND L.HD/=X THEN L.TL->L CLOSE; L.ISLINK;
END;



COMMENT 'RUNS THE PIECE OF PROGRAM IN LIST L';

FUNCTION EVAL L;
 POPVAL(L<>[GOON]);
END;



COMMENT 'RETURNS ITEM SUCCEEDING X IN L';

FUNCTION SUC L X;
 LOOPIF L.ISLINK AND L.HD/=X THEN L.TL.TL->L CLOSE;
 IF L.NULL THEN UNDEF ELSE L.TL.HD CLOSE;
END;
```

```
COMMENT 'RETURNS ITEM PRECEDING N IN L';

FUNCTION PRE L N;
 LOOPIF L.ISLINK AND L.TL.HD/=N THEN L.TL.TL->L CLOSE;
 IF L.NULL THEN UNDEF ELSE L.HD CLOSE;
END;



COMMENT 'DELETES N AND ITS SUCCESSOR FROM L';

FUNCTION DEL L N;
 IF NOT(MEMB(N,L)) THEN L EXIT;
 LOOPIF L.HD/=N THEN L.TL.TL<>[%L.HD,L.TL.HD%]->L CLOSE;
 L.TL.TL;
END;



COMMENT 'REPLACES SUCCESSOR OF N WITH X';

FUNCTION REP L N X;
 [%N,X%]<>DEL(L,N);
END;



COMMENT 'SUBSTITUTES X AFTER N IN LIST NAMED W';

FUNCTION SUB W N X;
 REP(W.VALOF,N,X)->W.VALOF;
END;



COMMENT 'EXCHANGES X2 FOR X1 AT ALL LEVELS OF LIST L';

FUNCTION XCH L X1 X2 => LL;
 NIL->LL; L.REV->L;
 LOOPIF L.ISLINK
 THEN IF L.HD.ISLIST THEN XCH(L.HD,X1,X2)
      ELSEIF L.HD=X1 THEN X2 ELSE L.HD
      CLOSE; ::LL->LL; L.TL->L;
 CLOSE;
END;



COMMENT 'THE FOLLOWING FUNCTIONS ARE USED FOR PRINTOUTS';



FUNCTION PRS X; X.PRSTRING; END;
```

```
FUNCTION PRL X;
 LOOPIF X.ISLINK THEN X.HD.PR; X.TL->X CLOSE;
END;


FUNCTION PRA L;
 1.NL;
 LOOPIF L.ISLINK THEN 2.SP;L.HD.PR;1.SP;L.TL.HD.PR;L.TL.TL->L CLOSE;
END;


FUNCTION PRC L;
 IF SUC(L,KIND)="GAME" THEN L.PRG ELSE L.PRF CLOSE;
END;


FUNCTION PRG L;
 2.NL; 'GAME `.PRS; SUC(L,NAME).PR;
 2.NL; 'CURRENT PLACE:`.PRS; SUC(L,PLACE).PR;
 2.NL; 'MY COLOUR: `.PRS; SUC(L,COLOUR).PR;
 2.NL; L.PRB;
END;


FUNCTION PRF L;
 VARS X;
 2.NL; 'ROUTINE `.PRS; SUC(L,NAME).PR;
 2.NL; 'CURRENT PLACE:`.PRS; SUC(L,PLACE).PR;
 2.NL; 'MARKED PLACE:`.PRS;
 IF ZEPLACE.NULL THEN 1.SP;"NONE"
 ELSEIF ZEPLACE.HD=SUC(L,NAME) THEN ZEPLACE.TL.HD ELSE 1.SP; "NONE"
 CLOSE; .PR;
 2.NL; L.PRB;
END;


FUNCTION PRB L;
 VARS X Y; SUC(L,NAME).VALOF->X; SUC(L,ENTRIES)->L; L.REV->L;
 'ENTRIES`.PRS;
 LOOPIF L.ISLINK
 THEN IF L.HD.ISLINK
      THEN 1.NL; 1.SP; L.TL.HD.PR; '. `.PRS; SUC(X,L.TL.HD).TL->Y;
           IF MEMB(Y.HD,[WHITE BLACK])
           THEN Y.TL.HD ELSE Y.HD
           CLOSE; .PR; 2.SP; L.HD.PR;
      CLOSE;
      L.TL.TL->L;
 CLOSE; 2.NL;
END;


FUNCTION PRW;
 1->PRW1; '(STATE OF WORLD IS NOW `.PRS; WOBJECTS.PR; ')`.PRS;
 IF PRO THEN 1.NL CLOSE;
END;
```

```
*** [WORLD] ***


VARS WMESSAGE WOBJECTS;

NIL->WMESSAGE;
[JOHN OUT MARY IN BOLT UP DOOR SHUT]->WOBJECTS;


COMMENT 'WOBJECTS HOLDS THE CURRENT STATE OF THE WORLD. WMESSAGE
         HOLDS UTTERANCES. THE FUNCTIONS WMOVE,WPUSH AND WSLIDE
         DEFINED BELOW REPRESENT THE ACTUAL LAWS OF THE UNIVERSE:
         THAT IS, THE THREE KINDS OF ACTION AND THEIR CONSEQUENCES';



FUNCTION WPOS X;
 SUC(WOBJECTS,X);
END;



FUNCTION WSUB N X;
 SUB("WOBJECTS",N,X);
END;



FUNCTION WMOVE ROBOT;
 IF WPOS(DOOR)=OPEN
 THEN IF WPOS(ROBOT)=IN
      THEN WSUB(ROBOT,OUT)
      ELSE WSUB(ROBOT,IN)
      CLOSE;
 CLOSE;
END;



FUNCTION WPUSH ROBOT;
 IF WPOS(BOLT)=UP
 THEN IF WPOS(DOOR)=OPEN
      THEN WSUB(DOOR,SHUT)
      ELSE WSUB(DOOR,OPEN)
      CLOSE;
 CLOSE;
END;



FUNCTION WSLIDE ROBOT;
 IF WPOS(ROBOT)=IN
 THEN IF WPOS(BOLT)=UP
      THEN WSUB(BOLT,DOWN)
      ELSE WSUB(BOLT,UP)
      CLOSE;
 CLOSE;
END;
```

```
*** [CONCEPTS] ***

COMMENT 'CONCEPTUAL KNOWLEDGE';

VARS ZCACTS ZCOBJS ZCTYPES ZCPROPS ZCTPROPS
     ZCWSHELL ZCFSHELL ZCGSHELL ZCRSHELL;



[MOVE SLIDE PUSH]->ZCACTS;
[JOHN MARY DOOR BOLT]->ZCOBJS;
[ROBOT DOOR BOLT]->ZCTYPES;
[IN OUT UP DOWN OPEN SHUT]->ZCPROPS;
[ROBOT[IN OUT] DOOR[OPEN SHUT] BOLT[UP DOWN]]->ZCTPROPS;



[JOHN UNDEF MARY UNDEF DOOR UNDEF BOLT UNDEF]->ZCWSHELL;

[KIND ROUTINE NAME UNDEF PLACE UNDEF ENTRIES UNDEF ]
->ZCFSHELL;

[KIND GAME NAME UNDEF PLACE UNDEF ENTRIES UNDEF COLOUR UNDEF]
->ZCGSHELL;

[[EVT[ROBOT PUSH] SIT[ANY] RES[UNDEF]]
 [EVT[ROBOT SLIDE] SIT[ANY] RES[UNDEF]]
 [EVT[ROBOT MOVE] SIT[ANY] RES[UNDEF]]]
->ZCRSHELL;



FUNCTION ZCTYPOF X;
 IF MEMB(X,ZCTYPES) THEN X ELSE "ROBOT" CLOSE;
END;



FUNCTION ZCPROPOF P X;
 MEMB(P,SUC(ZCTPROPS,X));
END;



FUNCTION ZCEVENT X;
 IF X.ISLIST AND X.LENGTH>1 AND MEMB(X.TL.HD,ZCACTS)
 THEN 1 ELSE 0 CLOSE;
END;



FUNCTION ZCGAME X;
 MEMB(X,[ZGPLAN ZGGOAL ZGGAME ZGRULE ZGASSESS ZGTELL ZGCHECK
 ZGASK]); END;
```

```
*** [KNOW1] ***

COMMENT 'EMPIRICAL KNOWLEDGE';


COMMENT 'VARIABLES SET BY OPERATOR BEFORE EACH RUN. THEY INDICATE
         THE ROBOTS NAME, THE NAME OF ITS PARTNER, WHAT IT CAN
         DO, WHAT IT CAN SEE, AND ITS MAIN GOAL';

VARS ZKME ZKYOU ZKACTS ZKSEE ZKGOAL;


COMMENT 'ROBOTS MODEL OF CURRENT WORLD SITUATION AND THE RULES
         BY WHICH THE WORLD WORKS. GAINED BY EXPERIENCE DURING
         THE RUN';

VARS ZKWORLD ZKRULES;


COMMENT 'ROBOTS MODEL OF PARTNERS MIND. INDICATES WHAT PARTNER
         CAN DO, WHAT IT CAN SEE, ITS THEORY OF HOW WORLD WORKS
         AND ITS GOAL';

VARS ZKXACTS ZKXSEE ZKXRULES ZKXGOAL;



COMMENT 'THE FIRST BATCH OF FUNCTIONS IS USED TO INITIALISE
         THE ABOVE VARIABLES';


COMMENT 'TAKES FRESH LOOK AT WHAT IT CAN SEE';

FUNCTION ZKLOOK;
 VARS V;
 ZKSEE->V;
 LOOPIF V.ISLINK THEN SUB("ZKWORLD",V.HD,WPOS(V.HD)); V.TL->V CLOSE;
END;



COMMENT 'PREPARES THE K VARIABLES AT START OF RUN';

FUNCTION ZKPREP;
 ZCWSHELL->ZKWORLD; .ZKLOOK;
 ZCRSHELL->ZKRULES;
 ZCWSHELL->ZKXSEE;
 NIL->ZKXGOAL; NIL->ZKXACTS;
 NIL->ZKXRULES;
END;
```

```
COMMENT 'THE SECOND BATCH OF FUNCTIONS IS CONCERNED WITH
         FINDING OUT WHETHER GOALS ARE DONE OR WHETHER
         THEY CAN BE DONE; SOMETIMES A FUNCTION SETS UP
         GAME ZGASK IN ORDER TO GET THE ANSWER`;



COMMENT 'FINDS TRUTH VALUE OF STATEMENT S IN WORLD W`;

FUNCTION ZKTVAL S W;
 VARS P; SUC(W,S.HD)->P;
 IF P=UNDEF THEN UNDEF ELSE P=S.TL.HD CLOSE;
END;



COMMENT 'FINDS OUT IF S IS THE CASE, USING OWN RESOURCES ONLY`;

FUNCTION ZKIS1 S;
 ZKTVAL(S,ZKWORLD);
END;



COMMENT 'FINDS OUT IF GOAL G CAN BE DONE; USES OWN RESOURCES ONLY`;

FUNCTION ZKCAN1 G;
 IF G.ZCEVENT
 THEN IF G.HD=ZKME
      THEN MEMB(G.TL.HD,ZKACTS)
      ELSE SUC(ZKXACTS,G.TL.HD)
      CLOSE;
 ELSE IF SUC(ZPSTATE, PRE(ZPGOAL,G))=[FAILED] THEN FALSE
      ELSE TRUE
      CLOSE;
 CLOSE;
END;



COMMENT 'FINDS OUT IF S IS THE CASE, ASKING IF NECESSARY
         UNLESS IT IS KNOWN THAT PARTNER ALSO DOESNT KNOW`;

FUNCTION ZKIS S;
 VARS A;
 S.ZKIS1->A;
 IF A/=UNDEF THEN A EXIT;
 IF SUC(ZKXSEE, S.HD)=0
 THEN UNDEF
 ELSE ZEPLAY("ZGASK", [IS]<>S); [ASKED]
 CLOSE;
END;
```

```
COMMENT 'FINDS OUT IF GOAL G CAN BE DONE, ASKING IF NECESSARY';

FUNCTION ZKCAN G;
 VARS A; G.ZKCAN1->A;
 IF A/=UNDEF THEN A ELSE ZEPLAY("ZGASK",[CAN]<>G); [ASKED] CLOSE;
END;



COMMENT 'FINDS OUT AND ENTERS STATE OF GOAL G, ASKING ANY
         NECESSARY QUESTIONS';

FUNCTION ZKSTATE G;
 VARS A;
 G.ZKIS->A;
 IF A=[ASKED] THEN RETURN
 ELSEIF A=UNDEF THEN [FAILED].ZAENTER; RETURN
 ELSEIF A=1 THEN [ACHIEVED].ZAENTER
 EXIT;
 G.ZKCAN->A;
 IF A=[ASKED] THEN RETURN
 ELSEIF A=0 THEN [FAILED].ZAENTER
 ELSE [NOTYET].ZAENTER
 CLOSE;
END;
```

```
*** [KNOW2] ***

COMMENT 'THE THIRD BATCH OF FUNCTIONS IS FOR ACCESSING AND
         UPDATING THE THEORY OF HOW THE WORLD WORKS, AND THE
         MODEL OF THE OTHER ROBOTS THEORY. THESE THEORIES ARE
         HELD IN ZKRULES AND ZKXRULES RESPECTIVELY';



COMMENT 'GIVEN LIST OF RULES L AND EVENT E, RETURNS
         RELEVANT RULE, OR UNDEF IF THERE IS NONE';

FUNCTION ZKRULE1 L E;
 LOOPIF L.ISLINK AND SUC(L.HD,EVT)/=([ROBOT]<>E.TL)
 THEN L.TL->L;
 CLOSE;
 IF L.NULL THEN UNDEF ELSE L.HD CLOSE;
END;



COMMENT 'GIVEN RULE R AND EVENT E, MAKES RULE SPECIFIC TO E';

FUNCTION ZKSPEC R E;
 XCH(R,"ROBOT",E.HD);
END;



COMMENT 'FINDS OWN RULE FOR EVENT E';

FUNCTION ZKRULE E;
 ZKRULE1(ZKRULES,E);
END;



COMMENT 'FINDS OWN SPECIFIC RULE FOR EVENT E';

FUNCTION ZKSRULE E;
 ZKSPEC(ZKRULE(E), E);
END;



COMMENT 'FINDS PARTNERS RULE FOR E, OR UNDEF IF NOT KNOWN';

FUNCTION ZKXRULE E;
 ZKRULE1(ZKXRULES, E);
END;
```

```
COMMENT 'FINDS PARTNERS SPECIFIC RULE FOR E, IF KNOWN,
         AND UNDEF IF NOT';

FUNCTION ZKXSRULE E => R;
 E.ZKXRULE->R;
 IF R/=UNDEF THEN ZKSPEC(R,E)->R CLOSE;
END;



COMMENT 'DELETES RULE FOR EVENT E FROM LIST L';

FUNCTION ZKDEL L E => LL;
 NIL->LL;
 LOOPIF L.ISLINK
 THEN IF SUC(L.HD, EVT)/=([ROBOT]<>E.TL)
      THEN LL<>[%L.HD%]->LL
      CLOSE; L.TL->L
 CLOSE;
END;



COMMENT 'GIVES INVERSE FORM OF RULE';

FUNCTION ZKINVERT R;
 VARS E S C;
 SUC(R,EVT)->E; SUC(R,SIT)->S; SUC(R,RES)->C;
 IF S=[ANY] THEN R EXIT;
 IF C=[NOTHING]
 THEN SUC([MOVE ROBOT SLIDE BOLT PUSH DOOR],E.TL.HD)::NIL
 ELSE [NOTHING]
 CLOSE; ->C;
 SUC(ZCTPROPS,S.HD)->R;
 IF S.TL.HD=R.HD THEN R.TL.HD ELSE R.HD CLOSE; ->R;
 [%EVT,E,SIT,[%S.HD,R%],RES,C%];
END;



COMMENT 'PUTS NEW RULE R IN LIST NAMED N';

FUNCTION ZKADD1 N R;
 IF SUC(R,SIT)/=[ANY] AND SUC(R,RES)=[NOTHING]
 THEN R.ZKINVERT->R
 CLOSE;
 R::(ZKDEL(N.VALOF, SUC(R, EVT)))->N.VALOF;
END;



COMMENT 'PUTS RULE R IN ROBOTS THEORY';

FUNCTION ZKADD R;
 ZKADD1("ZKRULES",R);
END;
```

```
COMMENT 'PUTS RULE R IN MODEL OF PARTNERS THEORY';

FUNCTION ZKXADD R;
 ZKADD1("ZKXRULES", R);
END;



COMMENT 'PREDICTED RESULT OF EVENT IN WORLD W BY SPECIFIC
         RULE R';

FUNCTION ZKPRED1 W R;
 VARS T S; SUC(R,SIT)->S;;
 IF S=[ANY] THEN SUC(R, RES) EXIT;
 ZKTVAL(S,W)->T;
 IF T=UNDEF THEN [UNDEF]<>S
 ELSEIF T=1 THEN SUC(R, RES) ELSE [NOTHING]
 CLOSE;
END;



COMMENT 'RESULT OF EVENT E IN WORLD W AS PREDICTED BY OWN RULES';

FUNCTION ZKPRED E W;
 ZKPRED1(W,E.ZKSRULE);
END;



COMMENT 'RESULT OF EVENT E IN WORLD W AS PREDICTED BY PARTNERS
         RULES IF KNOWN: UNDEF IF NOT. REALLY, OF COURSE, THE
         RESULT IS THAT PREDICTED BY MODEL OF PARTNERS THEORY';

FUNCTION ZKXPRED E W;
 VARS R; E.ZKXSRULE->R;
 IF R=UNDEF THEN UNDEF ELSE ZKPRED1(W,R) CLOSE;
END;



COMMENT 'RETURNS 0 IF ALL ACTIONS IN RULES PRODUCE NO RESULT
         AND 1 OTHERWISE';

FUNCTION ZKFLUID;
 VARS R; ZKRULES->R;
 LOOPIF R.ISLINK AND SUC(R.HD,RES)=[NOTHING] THEN R.TL->R CLOSE;
 R.ISLINK;
END;
```

```
COMMENT 'FINDS OPPOSITE OF SITUATION S';

FUNCTION ZKOPP S;
 VARS P;
 SUC(ZCTPROPS, S.HD.ZCTYPOF)->P;
 IF S.TL.HD=P.HD THEN P.TL.HD ELSE P.HD CLOSE; ::[%S.HD%]; .REV;
END;



COMMENT 'FINDS RULE TO ACHIEVE SPECIFIED RESULT C, RETURNING
         UNDEF IF NONE EXISTS';

FUNCTION ZKRES C;
 VARS T; ZKRULES->T;
 LOOPIF T.ISLINK AND SUC(T.HD, RES)/=C THEN T.TL->T CLOSE;
 IF T.NULL THEN UNDEF ELSE T.HD CLOSE;
END;



COMMENT 'CHANGES THEORY TO FIT EXPERIENCE. E IS THE EVENT WHICH
         HAS JUST OCCURRED AND C IS ITS CONSEQUENCE (OR RES).
         EE, CC, SS ARE THE EVT, RES AND SIT OF THE OLD RULE RR.
         THE FUNCTION ALTERS RR AND PUTS THE NEW RULE IN ZKRULES';

FUNCTION ZKPONDER E C;
 VARS EE CC SS RR L X;
 E.ZKRULE->RR;
 SUC(RR,EVT)->EE; SUC(RR,SIT)->SS; SUC(RR,RES)->CC;
 IF CC=[UNDEF] THEN REP(RR,RES,C).ZKADD; 1 EXIT;
 IF CC/=C AND C/=[NOTHING] AND CC/=[NOTHING]
 THEN REP(RR,RES,[%C.HD.ZCTYPOF%])->RR; REP(RR,SIT,[ANY]).ZKADD; 1
 EXIT;
 C->CC; REP(RR,RES,CC)->RR;
 ZCTYPES->L;
 LOOPIF L.HD=CC.HD OR L.HD=SS.HD THEN L.TL->L CLOSE;
 IF L.HD="ROBOT" THEN E.HD ELSE L.HD CLOSE; ->E;
 IF SUC(ZKWORLD,E)=UNDEF THEN ZEPLAY("ZGASK",E.ZAMAKEQ);[ASKED] EXIT;
 [%L.HD, SUC(ZKWORLD,E)%]->SS;
 REP(RR,SIT,SS).ZKADD; 1;
END;
```

```
COMMENT 'JUDGES WHETHER PLAN P WILL ACHIEVE GOAL G';

FUNCTION ZKJUDGE P G;
 VARS E S W;
 IF NOT(.ZKFLUID) THEN UNDEF EXIT;
 P.ZPEVT->E; P.ZPSIT->S;
 ZKWORLD->W;
 IF S/=UNDEF THEN REP(W,S.HD,S.TL.HD)->W CLOSE;
 ZKPRED(E,W)->S;
 IF S.HD=UNDEF
 THEN ZEPLAY("ZGASK",[IS]<>S.TL); [ASKED]
 ELSE S.HD=G.HD
 CLOSE;
END;



COMMENT 'RETURNS UNDEF IF R1 AND R2 ARE EQUALLY GOOD RULES,
         1 IF R1>R2, AND 0 IF R2>R1';

FUNCTION ZKBETTER R1 R2;
 IF R1=UNDEF THEN 0; RETURN ELSEIF R2=UNDEF THEN 1; EXIT;
 IF ZAEQUAL(R1,R2) THEN UNDEF EXIT;
 IF SUC(R1,RES)=[UNDEF] THEN 0 EXIT;
 IF SUC(R2,RES)=[UNDEF] THEN 1 EXIT;
 SUC(R1,SIT)/=[ANY] ->R1; SUC(R2,SIT)/=[ANY] ->R2;
 IF R1=R2 THEN UNDEF ELSE R2<R1 CLOSE;
END;
```

```
*** [PLAN] ***

VARS ZPCURR ZPGOAL ZPTREE ZPSTATE ZPACTOR ZPPLAN;



COMMENT 'RETURNS INDEX NUMBER OF NEXT GOAL TO ATTEMPT';

FUNCTION ZPNEXTGL => N1;
 VARS N2; [0]->N2;
 LOOPIF N2/=UNDEF THEN N2.HD->N1; SUC(ZPTREE,N1)->N2 CLOSE;
END;



COMMENT 'RETURNS PARENT OF NODE N IN ZPTREE';

FUNCTION ZPPARENT N;
 VARS T; ZPTREE->T;
 LOOPIF T.ISLINK AND NOT(MEMB(N,T.TL.HD)) THEN T.TL.TL->T CLOSE;
 IF T.NULL THEN UNDEF ELSE T.HD CLOSE;
END;



COMMENT 'HANGS PLAN P ON NODE N OF ZPTREE';

FUNCTION ZPHANG P N;
 VARS L Q;
 NIL->L;
 LOOPIF P.ISLINK
 THEN .ZAINDEX->Q; L<>[%Q%]->L;
      SUB("ZPACTOR",Q,P.HD); P.TL->P;
      SUB("ZPGOAL",Q,P.HD); P.TL->P;
      SUB("ZPSTATE",Q,NOTYET)
 CLOSE;
 SUB("ZPTREE",N,L);
END;



COMMENT 'DELETES ALL TREE BELOW NODE N';

FUNCTION ZPDELETE N;
 VARS S;
 ZPTREE,N.SUC->S;
 IF S=UNDEF THEN RETURN ELSE DEL(ZPTREE,N)->ZPTREE CLOSE;
 LOOPIF S.ISLINK THEN S.HD.ZPDELETE; S.TL->S CLOSE;
END;
```

```
COMMENT 'REMOVES TERMINAL NODE N FROM TREE';

FUNCTION ZPCHOP N;
 VARS P S; N.ZPPARENT->P; ZPTREE,P.SUC.TL->S;
 IF S.NULL THEN P.ZPDELETE ELSE SUB("ZPTREE",P,S) CLOSE;
END;



COMMENT 'BRINGS ABOUT EVENT E';

FUNCTION ZPDO E;
 E.HD; E.TL.HD->E;
 IF E=PUSH THEN .WPUSH
 ELSEIF E=SLIDE THEN .WSLIDE
 ELSEIF E=MOVE THEN .WMOVE
 CLOSE; 3.NL; .PRW;
END;



COMMENT 'FINDS THE EVENT IN PLAN P';

FUNCTION ZPEVT P;
 P.REV.HD;
END;



COMMENT 'FINDS THE SITN, IF ANY, IN PLAN P';

FUNCTION ZPSIT P;
 IF P.LENGTH=2 THEN UNDEF ELSE P.TL.HD CLOSE;
END;



COMMENT 'CLEARS AWAY INFORMATION ABOUT OLD GOALS';

FUNCTION ZPPRUNE;
 VARS T L P Q;
 NIL->L; ZPTREE->T;
 LOOPIF T.ISLINK THEN L<>T.TL.HD->L; T.TL.TL->T CLOSE;
 [ZPGOAL ZPACTOR ZPSTATE]->P;
L1:
 IF P.NULL THEN EXIT;
 P.HD.VALOF->T; NIL->Q;
 LOOPIF T.ISLINK THEN IF MEMB(T.HD,L) THEN [%T.HD,T.TL.HD%]
 <>Q->Q CLOSE; T.TL.TL->T CLOSE; Q->P.HD.VALOF; P.TL->P; GOTO L1;
END;
```

```
*** [ROUTINES] ***


ROUTINE ZRBASIC;
 1. ↑ ZRPREP;
 2. * ZRMAINGL;
 3. ↑ ZRPREP1;
 4. * ZRACHGL;
 5. ↑ ZRMOVE1;
 6. * ZRAPPEAL;
 7. ↑ ZRPREP2;
 8. * ZRACHGL;
 9. ↑ ZRHALT;
END;



ROUTINE ZRACHIEVE;
 1. * ZRCURRGL;
 2. * ZRACTOR;
 3. * ZRKIND;
 4. * ZRSTATE;
 5. * ZRPLAN1;
 6. ↑ ZRRETURN;
 7. * ZRBEFORE;
 8. * ZRACT;
 9. * ZRRESULT;
10. * ZRPARENT;
11. * ZRLESSON;
12. ↑ ZRPRUNE;
END;



ROUTINE ZRPLAN;
 1. * ZRGOAL;
 2. * ZRRULE;
 3. * ZRSPEC;
 4. * ZRSIT;
 5. ↑ ZRCOMP;
END;
```

```
*** [ROUTINE BASIC] ***


COMMENT '1. PREPARES WORLD MODEL (INITIALISES K VARIABLES)';

FUNCTION ZRPREP;
 .ZKPREP; ZAGOTO(2);
END;



COMMENT '2. ENTERS THE MAIN GOAL';

FUNCTION ZRMAINGL;
 ZKGOAL.ZAENTER;
END;



COMMENT '3. IF THERE IS A MAIN GOAL, PREPARES FOR SOLO ATTEMPT.
             IF NOT, HALTS';

FUNCTION ZRPREP1;
 VARS N;
 IF 2.ZAENTRY=[NONE] THEN ZAGOTO(9) EXIT;
 .ZAINDEX->N;
 NIL,NIL,NIL,NIL->ZPSTATE->ZPGOAL->ZPTREE->ZPACTOR;
 SUB("ZPGOAL",N,ZKGOAL);
 SUB("ZPTREE",0,[%N%]);
 SUB("ZPSTATE",N,NOTYET);
 SUB("ZPACTOR",N,ZKME);
 0->ZEJOINT; ZAGOTO(4);
END;



COMMENT '4 AND 8. ATTEMPTS GOAL, ENTERING [ACHIEVED] OR [FAILED]';

FUNCTION ZRACHGL;
 "ZRACHIEVE".ZELOAD;
END;



COMMENT '5. IF SOLO ATTEMPT SUCCEEDS, HALT; IF NOT, SEEK HELP';

FUNCTION ZRMOVE1;
 IF 4.ZAENTRY=[ACHIEVED] THEN ZAGOTO(9) ELSE ZAGOTO(6) CLOSE;
END;
```

```
COMMENT '6. ARRANGES GAME TO APPEAL FOR HELP. GAME ENTERS
            [YES] OR [NO]';

FUNCTION ZRAPPEAL;
 ZEPLAY("ZGGOAL",ZKGOAL);
END;



COMMENT '7. IF APPEAL SUCCEEDS, PREPARE FOR JOINT ATTEMPT.
            IF NOT, HALT';

FUNCTION ZRPREP2;
 VARS N;
 IF 6.ZAENTRY=[NO] AND NOT(ZEJOINT) THEN ZAGOTO(9) EXIT;
 .ZAINDEX->N;
 NIL,NIL,NIL,NIL->ZPGOAL->ZPTREE->ZPSTATE->ZPACTOR;
 SUB("ZPGOAL",N,ZKGOAL);
 SUB("ZPTREE",0,[%N%]);
 SUB("ZPSTATE",N,NOTYET);
 SUB("ZPACTOR",N,BOTH);
 1->ZEJOINT; ZAGOTO(8);
END;



COMMENT '9. KEEPS SWAPPING';

FUNCTION ZRHALT;
 [.ZESWAP]->ZENEXT;
END;
```

```
*** [ROUTINE ACHIEVE] ***

COMMENT '1. ENTERS NEXT GOAL TO BE TACKLED';

FUNCTION ZRCURRGL;
 VARS G;
 .ZPNEXTGL->ZPCURR;
 ZPGOAL.ZPCURR.SUC->G;
 G.ZAENTER;
END;



COMMENT '2. ENTERS ACTOR FOR CURRENT GOAL';

FUNCTION ZRACTOR;
 (ZPACTOR,ZPCURR.SUC::NIL).ZAENTER;
END;



COMMENT '3. ENTERS KIND OF GOAL: [EVENT] OR [SITN]';

FUNCTION ZRKIND;
 IF 1.ZAENTRY.ZCEVENT
 THEN [EVENT].ZAENTER; ZAGOTO(7)
 ELSE [SITN].ZAENTER
 CLOSE;
END;



COMMENT '4. ENTERS STATE OF GOAL. ASKS IF NECESSARY';

FUNCTION ZRSTATE;
 VARS A;
 1.ZAENTRY.ZKSTATE;
 4.ZAENTRY->A;
 IF A.ISLINK THEN SUB("ZPSTATE",ZPCURR,A.HD) CLOSE;
END;
```

```
COMMENT '5. SETS UP ROUTINE OR GAME TO FIND PLAN. ENTRY
           IS A PLAN OR [NO]';

FUNCTION ZRPLAN1;
 IF ZAENTRY(4)/=[NOTYET] THEN ZAGOTO(12) EXIT;
 IF ZAENTRY(2)=[BOTH]
 THEN ZEPLAY("ZGPLAN",UNDEF)
 ELSE "ZRPLAN".ZELOAD
 CLOSE;
END;



COMMENT '6. ADDS PLAN TO TREE AND RELOADS ROUTINE, OR FAILS
           THE GOAL IF NO PLAN WAS FOUND';

FUNCTION ZRRETURN;
 VARS P;
 5.ZAENTRY->P;
 IF P=[NO] THEN SUB("ZPSTATE",ZPCURR,FAILED); ZAGOTO(12) EXIT;
 ZPHANG(P,ZPCURR); .ZEEXIT; "ZRACHIEVE".ZELOAD;
END;



COMMENT '7. ENTERS CURRENT VALUE OF WORLD MODEL, FOR
           LATER COMPARISON';

FUNCTION ZRBEFORE;
 ZKWORLD.ZAENTER;
END;



COMMENT '8. WHOEVER IS TO DO THE ACTION TELLS THE OTHER
           WHEN IT IS DONE AND LOOKS AGAIN AT THE WORLD';

FUNCTION ZRACT;
 IF ZAENTRY(2)=[%ZKYOU%] THEN [.ZESWAP]->ZENEXT EXIT;
 1.ZAENTRY.ZPDO; [DONE].ZAENTER; SUB("ZPSTATE",ZPCURR,ACHIEVED);
 IF ZEJOINT THEN ZEPLAY("ZGTELL",1.ZAENTRY) CLOSE;
END;



COMMENT '9. ENTERS NAME OF OBJECT WHICH CHANGED POSITION,
           OR [NOTHING] IF NONE DID';


FUNCTION ZRRESULT;
 VARS D;
 IF ZEJOINT THEN ZEPLAY("ZGASSESS",UNDEF) EXIT;
 ZADIFF(ZAENTRY(7),ZKWORLD)->D;
 IF D.HD=UNDEF THEN ZEPLAY("ZGASK",D.TL.HD.ZAMAKEQ) EXIT;
 D.ZAENTER;
END;
```

```
COMMENT '10. FINDS STATE OF PARENT GOAL, AND PUTS STATE ON TREE';

FUNCTION ZRPARENT;
 VARS A N;
 ZPCURR.ZPPARENT->N;
 ZPGOAL,N.SUC.ZKSTATE;
 10.ZAENTRY->A;
 IF A.ISLINK THEN SUB("ZPSTATE",N,A.HD) CLOSE;
END;



COMMENT '11. LEARNS LESSON FROM SEEING RESULT OF EVENT. IF
             PARTNER IS KNOWN TO BELIEVE A RULE WHICH PREDICTS
             THE WRONG RESULT, TEACHES HIM BETTER RULE';

FUNCTION ZRLESSON;
 VARS P;
 IF ZKPRED(1.ZAENTRY,7.ZAENTRY)/=ZAENTRY(9)
 THEN ZKPONDER(1.ZAENTRY,9.ZAENTRY)->P; IF P=[ASKED] THEN EXIT;
 CLOSE;
 [LEARNED].ZAENTER;
 ZKXPRED(1.ZAENTRY,7.ZAENTRY)->P;
 IF P/=UNDEF AND P/=ZAENTRY(9)
 THEN ZEPLAY("ZGRULE",1.ZAENTRY.ZKRULE)
 CLOSE;
END;



COMMENT '12. PRUNES TREE READY FOR RELOADING ROUTINE. WIPES OFF
             ANY ACHIEVED OR FAILED GOALS, AND RETURNS TO
             ZRBASIC IF THE MAIN GOAL IS AMONG THEM';

FUNCTION ZRPRUNE;
 VARS S P;
 SUC(ZPSTATE,HD(SUC(ZPTREE,0)))->S;
 IF S/=NOTYET THEN .ZEEXIT; [%S%].ZAENTER EXIT;
 10.ZAENTRY->S; ZPCURR.ZPPARENT->P;
 IF S=[ACHIEVED] THEN P.ZPDELETE; P.ZPCHOP
 ELSEIF S=[FAILED] THEN P.ZPPARENT.ZPDELETE
 ELSE ZPCURR.ZPCHOP
 CLOSE;
 .ZPPRUNE; .ZEEXIT; "ZRACHIEVE".ZELOAD;
END;
```

```
*** [ROUTINE PLAN] ***

COMMENT '1. ENTERS GOAL FOR WHICH PLAN IS NEEDED';

FUNCTION ZRGOAL;
 ZPGOAL,ZPCURR.SUC.ZAENTER;
END;



COMMENT '2. FINDS RULE TO ACHIEVE REQUIRED RESULT: IF NONE, FAILS';

FUNCTION ZRRULE;
 VARS R;
 IF NOT(.ZKFLUID) THEN .ZEEXIT; [NO].ZAENTER EXIT;
 [%1.ZAENTRY.HD.ZCTYPOF%].ZKRES->R;
 IF R=UNDEF THEN [UNDEF].ZKRES->R CLOSE;
 IF R=UNDEF THEN [NOTHING].ZKRES->R CLOSE;
 R.ZAENTER; ZKDEL(ZKRULES,SUC(R,EVT))<>[%R%]->ZKRULES;
END;



COMMENT '3. DECIDES WHO DOES THE ACTION, AND ENTERS A SPECIFIC RULE';

FUNCTION ZRSPEC;
 VARS R E A;
 2.ZAENTRY->R; ZKME::(SUC(R,EVT).TL)->E;
 IF SUC(R,RES)=[ROBOT] THEN (1.ZAENTRY.HD::E.TL)->E CLOSE;
 IF E.HD=ZKME THEN GOTO ME ELSE GOTO HIM CLOSE;
ME: IF E.ZKCAN THEN ZKSPEC(R,E).ZAENTER EXIT;
HIM: IF ZEJOINT
 THEN ZKYOU::E.TL->E; E.ZKCAN->A;
      IF A=[ASKED] THEN RETURN
      ELSEIF A=1 THEN ZKSPEC(R,E).ZAENTER;
      EXIT;
 CLOSE;
 .ZEEXIT; [NO].ZAENTER;
END;



COMMENT '4. ENTERS THE STATE OF THE SIT, ASKING IF NECESSARY';

FUNCTION ZRSIT;
 VARS S; SUC(3.ZAENTRY, SIT)->S;
 IF S=[ANY] THEN [ACHIEVED].ZAENTER ELSE S.ZKSTATE CLOSE;
END;
```

```
COMMENT '5. COMPLETES PLAN AND ENTERS IN THE CALLING F/G';

FUNCTION ZRCOMP;
 VARS P R;
 IF 4.ZAENTRY=[FAILED] THEN [NO]->P; GOTO LAST CLOSE;
 3.ZAENTRY->R; NIL->P;
 IF 4.ZAENTRY=[NOTYET]
 THEN [%SUC(R,SIT)%]->P;
 IF ZEJOINT THEN BOTH ELSE ZKME CLOSE; ::P->P;
 CLOSE;
 SUC(R,EVT)->R; P<>[%R.HD,R%]->P;
LAST:
 .ZEEXIT; P.ZAENTER;
END;
```

```
*** [GAMES] ***


GAME ZGGAME;
 1. * W ZGNAME [ZDGAME];
 2. * B ZGREADY [ZDSIGN];
 3. ↑ W ZGLOAD;
END;


GAME ZGASK;
 1. * W ZGQUERY [ZDQUERY];
 2. * B ZGANSWER [ZDSIGN];
 3. ↑ W ZGRECORD;
END;


GAME ZGTELL;
 1. * W ZGRELATE [ZDFACT ZDEVENT];
 2. * B ZGEXAMINE [ZDSIGN];
 3. ↑ W ZGRELOOK;
END;


GAME ZGRULE;
 1. * W ZGWRULE [ZDRULE];
 2. * B ZGBREPLY [ZDSIGN];
 3. ↑ W ZGWNOTE;
 4. * B ZGBRULE [ZDRULE];
 5. * W ZGWREPLY [ZDSIGN];
 6. ↑ B ZGBNOTE;
END;


GAME ZGGOAL;
 1. * W ZGPLEAD [ZDSITN];
 2. * B ZGREACT [ZDSIGN];
 3. ↑ W ZGREPORT;
END;


GAME ZGPLAN;
 1. * W ZGSUGGEST [ZDPLAN ZDSIGN];
 2. * B ZGRESPOND [ZDSIGN];
 3. ↑ W ZGRETURN;
END;


GAME ZGASSESS;
 1. * W ZGCHANGE [ZDOBJ];
 2. * B ZGCONFIRM [ZDSIGN];
 3. * W ZGPARENT [ZDFACT];
 4. * B ZGBENTER [ZDSIGN];
 5. ↑ W ZGWENTER;
END;
```

```
*** [GAME GAME] ***

COMMENT 'USED TO GET ANOTHER GAME LOADED. CAN BE USED ANYWHERE';



COMMENT 'WHITE NAMES THE GAME';

FUNCTION ZGNAME;
 ZEGAME1.ZAENTER;
END;



COMMENT 'BLACK AGREES TO PLAY, POSSIBLY AFTER A DELAY WHILE HE
         REACHES A SUITABLE POINT IN HIS ROUTINES, LOADS THE
         GAME TAKING BLACK, AND EXITS FROM ZGGAME';

FUNCTION ZGREADY;
 VARS G; 1.ZAENTRY->G;
 IF G=[JUMP] THEN G.ZAENTER; .ZEJUMP EXIT;
 IF G=[ZGGOAL] AND ZATAKE1("ZRBASIC",PLACE)=1
 THEN [ZRBASIC 3].ZEMARK; RETURN
 ELSEIF G=[ZGPLAN] AND ZATAKE1("ZRACHIEVE",PLACE)/=5
 THEN [ZRACHIEVE 5].ZEMARK; RETURN
 CLOSE;
 IF MEMB(G.HD,[ZGRULE ZGTELL])
 AND SUC(ZECONTROL.TL.HD,KIND)="GAME"
 THEN ZAWIPE(SUC(ZECONTROL.TL.HD,NAME))
 CLOSE;
 [YES].ZAENTER; .ZEEXIT; G.HD.ZELOAD; ZAPUT(COLOUR,BLACK);
END;


COMMENT 'WHITE LOADS THE GAME, TAKING WHITE, AND EXITS';
FUNCTION ZGLOAD;
 VARS G; 1.ZAENTRY->G;
 IF G=[JUMP] THEN .ZEJUMP ELSE .ZEEXIT; G.HD.ZELOAD CLOSE;
END;
```

```
*** [GAME ASK] ***

COMMENT 'USED ANYWHERE. ASKS EITHER "CAN" OR "IS" QUESTIONS.
         LEARNS NOT ONLY THE ANSWER, BUT WHETHER OR NOT THE
         OTHER ROBOT KNOWS THE ANSWER';



COMMENT '1. ASKS QUESTION';

FUNCTION ZGQUERY;
 ZEMOVE1.ZAENTER;
END;



COMMENT '2. TRIES TO FIND ANSWER, USING OWN RESOURCES ONLY.
            IF IT IS AN "IS" QUESTION, RECORDS THE FACT THAT
            WHITE DIDNT KNOW THE ANSWER';

FUNCTION ZGANSWER;
 VARS Q; 1.ZAENTRY->Q;
 IF Q.HD="IS"
 THEN SUB("ZKXSEE",Q.TL.HD,0); Q.TL.ZKIS1
 ELSE Q.TL.ZKCAN1
 CLOSE; ->Q; [%Q.ZOYN%].ZAENTER;
 .ZEEXIT;
END;



COMMENT '3. RECORDS THE ANSWER AND WHETHER BLACK KNEW IT';

FUNCTION ZGRECORD;
 VARS Q A; 1.ZAENTRY->Q; 2.ZAENTRY->A;
 IF A=[UNDEF] THEN UNDEF ELSE (A=[YES]) CLOSE; ->A;
 IF Q.HD="CAN" THEN SUB("ZKXACTS",Q.TL.TL.HD,A); .ZEEXIT EXIT;
 Q.TL->Q; SUB("ZKXSEE",Q.HD,(A/=UNDEF));
 IF A=0 THEN Q.ZKOPP->Q CLOSE;
 IF A/=UNDEF THEN SUB("ZKWORLD",Q.HD,Q.TL.HD) CLOSE;
 .ZEEXIT;
END;
```

```
*** [GAME TELL] ***

COMMENT 'USED FOR WHITE TO TELL BLACK SOMETHING. CAN BE PLAYED
         ANYWHERE';



COMMENT 'WHITE ENTERS A FACT';

FUNCTION ZGRELATE;
 ZEMOVE1.ZAENTER;
END;



COMMENT 'BLACK UPDATES HIS MODELS OF OBJECT POSITIONS, AND OF WHITE,
         AND RESPONDS TO WHAT WHITE TOLD HIM';

FUNCTION ZGEXAMINE;
 VARS F; 1.ZAENTRY->F;
 IF F.ZCEVENT
 THEN IF ZATAKE1("ZRACHIEVE",PLACE)=8
      THEN [UNDEF].ZAENTER; .ZEEXIT; [DONE].ZAENTER;
           SUB("ZKXACTS",F.TL.HD,1); SUB("ZPSTATE",ZPCURR,ACHIEVED);
      ELSE [ZRACHIEVE 8].ZEMARK
      CLOSE;
 EXIT;
 IF F.TL.HD="IS"
 THEN IF F.HD=1 THEN F.TL.TL ELSE F.TL.TL.ZKOPP CLOSE; ->F;
      IF MEMB(F.HD,ZKSEE)
      THEN IF F.ZKIS1
           THEN [YES]; SUB("ZKXSEE",F.HD,1)
           ELSE [NO]; SUB("ZKXSEE",F.HD,0)
           CLOSE;
      ELSE [UNDEF]; SUB("ZKXSEE",F.HD,1);SUB("ZKWORLD",F.HD,F.TL.HD)
      CLOSE;
 ELSE IF F.TL.TL.HD=ZKME
      THEN IF MEMB(F.TL.TL.TL.HD,ZKACTS)=F.HD
           THEN [YES]
           ELSE [NO]
           CLOSE;
      ELSE [UNDEF]; SUB("ZKXACTS",F.TL.TL.TL.HD,F.HD)
      CLOSE;
 CLOSE;
 .ZAENTER; .ZEEXIT;
END;
```

```
COMMENT 'WHITE UPDATES HIS MODELS OF OBJECT POSITIONS, AND OF BLAC
        IN THE LIGHT OF BLACKS RESPONSE';
FUNCTION ZGRELOOK;
 VARS F A; 1.ZAENTRY->F; 2.ZAENTRY->A;
 IF F.ZCEVENT THEN .ZEEXIT EXIT;
 IF F.TL.HD="IS"
 THEN IF A=[NO]
      THEN SUB("ZKXSEE",F.TL.TL.HD,NOT(MEMB(F.TL.TL.HD,ZKSEE)));
           IF SUC(ZKXSEE,F.TL.TL.HD)
           THEN IF F.HD=0 THEN F.TL.TL ELSE F.TL.TL.ZKOPP CLOSE;
                ->F; SUB("ZKWORLD",F.HD,F.TL.HD)
           CLOSE;
      ELSE SUB("ZKXSEE",F.TL.TL.HD,(A=[YES]))
      CLOSE;
 ELSE IF F.TL.TL.HD=ZKYOU AND A=[NO]
      THEN SUB("ZKXACTS",F.TL.TL.TL.HD,F.HD.NOT)
      CLOSE;
 CLOSE;
 .ZEEXIT;
END;
```

```
*** [GAME RULE] ***

COMMENT 'USED TO EXPLAIN, OR COMPARE VIEWS ON, THE RULES WHICH
         SPECIFY THE CONSEQUENCES OF ACTIONS. CAN BE PLAYED ANYWHERE';



COMMENT 'WHITE ANNOUNCES A RULE';

FUNCTION ZGWRULE;
 ZEMOVE1.ZAENTER;
END;



COMMENT 'BLACK AGREES OR DISAGREES, EXITING IF HE AGREES';

FUNCTION ZGBREPLY;
 VARS R1 R2 B;
 1.ZAENTRY->R1; R1,EVT.SUC.ZKRULE->R2;
 R1.ZKXADD; ZKBETTER(R1,R2)->B;
 IF ZAEQUAL(R1,R2) THEN [YES].ZAENTER; .ZEEXIT
 ELSEIF B=1 THEN R1.ZKADD; [UNDEF].ZAENTER; .ZEEXIT
 ELSE [NO].ZAENTER
 CLOSE;
END;



COMMENT 'IF BLACK AGREED, WHITE UPDATES HIS MODEL OF BLACKS RULES
         AND EXITS. IF NOT, HE GOES TO 4 AND ALLOWS BLACK TO
         ANNOUNCE HIS RULE';

FUNCTION ZGWNOTE;
 IF 2.ZAENTRY=[NO] THEN 4.ZAGOTO ELSE 1.ZAENTRY.ZKXADD;.ZEEXIT CLOSE;
END;



COMMENT 'BLACK ANNOUNCES HIS VERSION OF THE RULE WHITE
         ANNOUNCED AT 1';

FUNCTION ZGBRULE;
 1.ZAENTRY,EVT.SUC.ZKRULE.ZAENTER;
END;
```

```
COMMENT 'WHITE RESPONDS, UPDATES HIS WORLD MODEL IN A WAY
         DEPENDING ON WHOSE RULE WAS BETTER, AND EXITS';

FUNCTION ZGWREPLY;
 VARS R1 R2 B;
 4.ZAENTRY->R1; 1.ZAENTRY->R2;
 R1.ZKXADD; ZKBETTER(R1,R2)->B; IF B=1 THEN R1.ZKADD CLOSE;
 IF B=UNDEF AND NOT(ZAEQUAL(R1,R2)) THEN 0->B CLOSE;
 IF B=UNDEF THEN [YES] ELSEIF B THEN [UNDEF] ELSE [NO] CLOSE;
 .ZAENTER; .ZEEXIT;
END;



COMMENT 'BLACK UPDATES HIS MODEL OF WHITES RULES, THEN EXITS';

FUNCTION ZGBNOTE;
 IF 5.ZAENTRY=[UNDEF] THEN 4.ZAENTRY.ZKXADD CLOSE; .ZEEXIT;
END;
```

```
*** [GAME GOAL] ***

COMMENT 'PLAYED WITH WHITE AT ZRBASIC 6 AND BLACK ANYWHERE';



COMMENT '1. WHITE ANNOUNCES HIS GOAL';

FUNCTION ZGPLEAD;
 ZEMOVE1.ZAENTER;
END;



COMMENT '2. IF BLACK HAS A DIFFERENT GOAL HE REFUSES; IF NOT,
            HE ACCEPTS AND GOES TO ZRBASIC 7';

FUNCTION ZGREACT;
 1.ZAENTRY->ZKXGOAL;
 IF ZKGOAL=[NONE] THEN ZKXGOAL->ZKGOAL CLOSE;
 IF ZKGOAL=ZKXGOAL THEN [YES] ELSE [NO] CLOSE; .ZAENTER;
 IF ZKGOAL=ZKXGOAL THEN 1->ZEJOINT; .ZEJUMP EXIT;
 .ZEEXIT;
END;



COMMENT '3. UPDATES MODEL OF PARTNERS GOAL, AND ENTERS HIS REPLY
            IN ZRBASIC 6';

FUNCTION ZGREPORT;
 VARS R; 2.ZAENTRY->R;
 IF R=[YES] THEN ZKGOAL->ZKXGOAL; 1->ZEJOINT CLOSE;
 .ZEEXIT; R.ZAENTER;
END;
```

```
*** [GAME PLAN] ***

COMMENT 'PLAYED WITH BOTH ROBOTS AT ZRACHIEVE 5. USED TO
         AGREE ON A PLAN';



COMMENT 'WHITE LOADS ZRPLAN, WHICH MAKES AND ENTERS THE PLAN';

FUNCTION ZGSUGGEST;
 "ZRPLAN".ZELOAD;
END;



COMMENT 'BLACK JUDGES THE PLAN, AND EITHER AGREES WITH IT AND ENTERS
         IT IN ZRACHIEVE 5, OR ARRANGES ZGRULE TO EXPLAIN THE RULE
         BY WHICH HE REJECTS IT';

FUNCTION ZGRESPOND;
 VARS P A;
 1.ZAENTRY->P;
 IF P=[NO]
 THEN IF .ZKFLUID
      THEN [YES].ZAENTER; .ZEEXIT; "ZGPLAN".ZELOAD
      ELSE [NO].ZAENTER; .ZEEXIT; [NO].ZAENTER; 6.ZAGOTO
      CLOSE;
 ELSE IF P.ZPEVT.ZKCAN1=0
      THEN ZEPLAY("ZGTELL",[0 CAN]<>P.ZPEVT); "ZGPLAN".ZAWIPE
      EXIT;
      ZKJUDGE(P,SUC(ZPGOAL,ZPCURR))->A; IF A=[ASKED] THEN EXIT;
      IF A=0 AND ZKBETTER(P.ZPEVT.ZKRULE,P.ZPEVT.ZKXRULE)=1
      THEN P.ZPEVT.ZKRULE->A;
           IF SUC(A,RES)/=[NOTHING] THEN A.ZKINVERT->A CLOSE;
           ZEPLAY("ZGRULE",A); "ZGPLAN".ZAWIPE
      ELSE [YES].ZAENTER; .ZEEXIT; P.ZAENTER; 6.ZAGOTO
      CLOSE;
 CLOSE;
END;



COMMENT 'WHITE MAKES AN ENTRY IN ZRACHIEVE 5, AND EXITS';

FUNCTION ZGRETURN;
 VARS P; 1.ZAENTRY->P;
 IF P=[NO] AND 2.ZAENTRY=[YES]
 THEN .ZEEXIT; "ZGPLAN".ZELOAD; ZAPUT(COLOUR,BLACK)
 ELSE .ZEEXIT; P.ZAENTER; 6.ZAGOTO
 CLOSE;
END;
```

```
*** [GAME ASSESS] ***

COMMENT 'PLAYED WITH THE ROBOTS AT ZRACHIEVE 8 OR 9. USED TO
         ASSESS THE RESULT OF AN ACTION';



COMMENT 'WHITE ENTERS ANY CHANGE HE NOTICES IN THE WORLD';

FUNCTION ZGCHANGE;
 VARS D; .ZKLOOK; ZADIFF(ZAENTRY1("ZRACHIEVE",7),ZKWORLD)->D;
 IF D.HD=UNDEF THEN ZEPLAY("ZGASK",D.TL.HD.ZAMAKEQ) EXIT;
 D.ZAENTER;
END;



COMMENT 'BLACK REACTS TO WHITES OBSERVATION, EITHER AGREEING,
         OR ARRANGING A GAME TO TELL WHITE WHAT HE OBVIOUSLY
         NEEDS TO KNOW';

FUNCTION ZGCONFIRM;
 VARS D E; 1.ZAENTRY->E; .ZKLOOK;
 ZADIFF(ZAENTRY1("ZRACHIEVE",7),ZKWORLD)->D;
 IF D=E OR D.HD=UNDEF
 THEN [YES].ZAENTER;
 ELSE IF E=[NOTHING] THEN D.HD ELSE E.HD CLOSE; ->E;
      ZEPLAY("ZGTELL",[1 IS]<>[%E,SUC(ZKWORLD,E)%]);
      "ZGASSESS".ZAWIPE;
 CLOSE;
END;



COMMENT 'WHITE ANNOUNCES THE STATE OF THE GOAL WHICH THE ACTION
         WAS MEANT TO ACHIEVE';

FUNCTION ZGPARENT;
 VARS N S G;
 ZPCURR.ZPPARENT->N; ZPGOAL,N.SUC->G; G.ZKIS->S;
 IF S=[ASKED] THEN EXIT;
 ZAENTER(S::[IS]<>G);
 IF S THEN [ACHIEVED] ELSE [NOTYET] CLOSE; ->S;
END;
```

```
COMMENT 'IF BLACK AGREES, HE MAKES ENTRIES 9 AND 10 IN ZRACHIEVE
         AND EXITS; IF NOT, HE ARRANGES ZGTELL TO TELL WHITE
         WHAT IS REALLY THE CASE';

FUNCTION ZGBENTER;
 VARS F S C;
 3.ZAENTRY->F; F.TL.TL.ZKIS1->S; 1.ZAENTRY->C;
 IF S=UNDEF OR S=F.HD
 THEN IF F.HD THEN [ACHIEVED] ELSE [NOTYET] CLOSE;->S;[YES].ZAENTER;
      .ZEEXIT; ZAGOTO(9); C.ZAENTER; ZAGOTO(10); S.ZAENTER;
      SUB("ZPSTATE",ZPCURR.ZPPARENT,S.HD);
 ELSE F.TL.TL.HD->F; ZEPLAY("ZGTELL",[1 IS]<>[%F,SUC(ZKWORLD,F)%]);
      "ZGASSESS".ZAWIPE
 CLOSE;
END;



COMMENT 'WHITE MAKES ENTRIES 9 AND 10 IN ZRACHIEVE';

FUNCTION ZGWENTER;
 VARS C S; 1.ZAENTRY->C; 3.ZAENTRY->S;
 .ZEEXIT; ZAGOTO(9); C.ZAENTER; ZAGOTO(10);
 IF S.HD=1 THEN [ACHIEVED] ELSE [NOTYET] CLOSE; ->S;
 S.ZAENTER; SUB("ZPSTATE",ZPCURR.ZPPARENT,S.HD);
END;
```

```
*** [AUXFUNS] ***


COMMENT 'PUTS X AFTER N IN ROUTINE/GAME W IN CONTROL';

FUNCTION ZAPUT1 W N X;
 VARS L1 L2 L3;
 ZECONTROL->L2;
 NIL->L1;
LOOP:
 IF L2.NULL THEN EXIT;
 L2.HD->L3;
 IF SUC(L3,NAME)=W
 THEN REP(L3,N,X)->L3; L1.REV<>(L3::L2.TL)->ZECONTROL
 ELSE L3::L1->L1; L2.TL->L2; GOTO LOOP
 CLOSE;
END;



COMMENT 'FINDS ITEM AFTER N IN ROUTINE/GAME W IN CONTROL';

FUNCTION ZATAKE1 W N;
 VARS L1;
 ZECONTROL->L1;
 LOOPIF L1.ISLINK AND SUC(L1.HD,NAME)/=W THEN L1.TL->L1 CLOSE;
 IF L1.NULL THEN UNDEF ELSE SUC(L1.HD,N) CLOSE;
END;



COMMENT 'PUTS IN X AT N IN ENTRIES OF W';

FUNCTION ZAENTER1 W N X;
 ZAPUT1(W,ENTRIES,REP(ZATAKE1(W,ENTRIES),N,X));
END;



COMMENT 'FINDS ENTRY AT N IN ROUTINE/GAME W';

FUNCTION ZAENTRY1 W N;
 SUC(ZATAKE1(W,ENTRIES),N);
END;



COMMENT 'PUTS X AFTER N IN TOP R/G OF CONTROL';

FUNCTION ZAPUT N X;
 ZAPUT1(NAME.ZATAKE,N,X);
END;
```

```
COMMENT 'FINDS ITEM AFTER N IN TOP R/G OF CONTROL';

FUNCTION ZATAKE N;
 SUC(ZECONTROL.HD,N);
END;



COMMENT 'ENTERS X AT CURRENT PLACE IN TOP R/G';

FUNCTION ZAENTER X;
 IF PRO THEN .ZOSP;PLACE.ZATAKE.PR;1.SP;X.PR;
 ' ENTERED BY '.PRS; ZKME.PR CLOSE;
 ZAENTER1(ZATAKE(NAME),ZATAKE(PLACE),X); X.ZAUTTER;
END;



COMMENT 'FINDS ENTRY AT PLACE N IN TOP R/G';

FUNCTION ZAENTRY N;
 ZAENTRY1(NAME.ZATAKE,N);
END;



COMMENT 'KILLS ANY R/G NAMED W';

FUNCTION ZAKILL W;
 VARS L1; NIL->L1;
 LOOPIF ZECONTROL.ISLINK AND ZATAKE(NAME)/=W
 THEN ZECONTROL.HD::L1->L1; ZECONTROL.TL->ZECONTROL
 CLOSE;
 IF ZECONTROL.ISLINK THEN ZECONTROL.TL->ZECONTROL CLOSE;
 L1.REV<>ZECONTROL->ZECONTROL;
END;



COMMENT 'WORKS OUT LIST OF EMPTY ENTRIES FOR R/G W';

FUNCTION ZAENTS W => L;
 NIL->L; W.VALOF->W;
 LOOPIF W.ISLINK
 THEN IF W.TL.HD.HD="*"
      THEN W.HD::(NIL::L)->L
      CLOSE;
      W.TL.TL->W;
 CLOSE;
END;
```

```
COMMENT 'USED TO MOVE PLACE IN A R/G';

FUNCTION ZAGOTO N;
 ZAPUT(PLACE,N);
END;



COMMENT 'PROVIDES NEW NUMBERS FOR USE AS INDEXES';


VARS ZACOUNT; 1->ZACOUNT;
FUNCTION ZAINDEX;
 ZACOUNT; ZACOUNT+1->ZACOUNT;
END;



COMMENT 'COMPARES TWO WORLD SITUATIONS AND RETURNS THE FACTOR
         ON WHICH THEY DIFFER, OR [NOTHING] IF THEY ARE THE SAME';

FUNCTION ZADIFF L1 L2;
 LOOPIF L1.ISLINK
 THEN IF L1.TL.HD/=SUC(L2,L1.HD)
      THEN IF SUC(L2,L1.HD)=UNDEF
           THEN [%UNDEF,L1.HD%]
           ELSE [%L1.HD%]
           CLOSE; RETURN
      ELSE L1.TL.TL->L1
      CLOSE;
 CLOSE; [NOTHING]
END;



COMMENT 'JUMPS TO R/G X, KILLING PROCEDURES ABOVE X IN CONTROL';

FUNCTION ZAJUMP X;
 LOOPIF ZECONTROL.LENGTH>1 AND ZATAKE(NAME)/=X
 THEN ZECONTROL.TL->ZECONTROL
 CLOSE;
END;



COMMENT 'TESTS TWO ASSOCIATION LISTS FOR EQUALITY';

FUNCTION ZAEQUAL L1 L2;
 ZADIFF(L1,L2)=[NOTHING];
END;
```

```
COMMENT 'MAKES A QUESTION TO DISCOVER THE POSITION OF X';

FUNCTION ZAMAKEQ X;
 [IS]<>[%X,ZCTPROPS,X.ZCTYPOF.SUC.HD%];
END;



COMMENT 'RETURNS 1 IF THERE IS NO GAME IN PROGRESS AND 0 OTHERWISE';

FUNCTION ZANOGAME;
 VARS L; ZECONTROL->L;
 LOOPIF L.ISLINK
 THEN IF SUC(L.HD,KIND)="GAME" THEN 0 EXIT; L.TL->L;
 CLOSE; ZEPLACE.NULL;
END;



COMMENT 'REMOVES ENTRIES IN STRUCTURE X AND BEGINS AT 1';

FUNCTION ZAWIPE X;
 ZAPUT1(X,PLACE,1);
 ZAPUT1(X,ENTRIES,X.ZAENTS);
END;



COMMENT 'ARRANGES FOR ENTRY X TO BE TRANSLATED INTO ENGLISH,
         THEN POSTS IT IN ZEBOX';

FUNCTION ZAUTTER X;
 VARS N;
 IF ZATAKE(KIND)="ROUTINE" THEN EXIT;
 ZATAKE(NAME).VALOF->N;
 IF (N,PLACE.ZATAKE.SUC.TL.HD/=COLOUR.ZATAKE) THEN EXIT;
 [%N,PLACE.ZATAKE.SUC.TL.TL.HD.DESTWORD%]->N; (N.HD::(55::N.TL.TL))->N;
 LOOPIF N.ISLINK THEN N.HD; N.TL->N CLOSE; .CONSWORD->N;
 [%X,".",N%].EVAL.ZEPOST;
END;
```

```
*** [EXEC] ***

COMMENT 'EXECUTIVE VARIABLES AND FUNCTIONS WHICH INTERPRET
         THE ROUTINES AND GAMES';



VARS ZECONTROL ZEBOX ZEAGAIN ZENEXT ZEJOINT ZEGAME1 ZEMOVE1
     ZEHOLD ZEPLACE;



COMMENT 'MASTER FUNCTION USED BY CHAIRMAN TO AROUSE ROBOT';

FUNCTION ZEAROUSE;
 IF .ZECALLED THEN .ZEREADY CLOSE; .ZEEXEC;
 LOOPIF ZEAGAIN THEN .ZEEXEC CLOSE;
END;



COMMENT 'RUNS EITHER ZECALL, ZESEND, ZESWAP, OR ZECONT, DEPENDING
         ON WHICH WAS LAST PUT INTO ZENEXT';

FUNCTION ZEEXEC;
 ZENEXT.EVAL;
END;



COMMENT 'CALLS THE OTHER ROBOT BY NAME';

FUNCTION ZECALL;
 [%ZKYOU%].ZEPOST;
 "ZGGAME".ZELOAD;
END;



COMMENT 'MAKES AN UTTERANCE';

FUNCTION ZESEND;
 0->ZEAGAIN;
 ZEBOX->WMESSAGE;
 UNDEF->ZEBOX;
 [.ZECONT]->ZENEXT;
END;
```

```
COMMENT 'RETURNS CONTROL TO THE CHAIRMAN WITHOUT SAYING ANYTHING';

FUNCTION ZESWAP;
 0->ZEAGAIN;
 NIL->WMESSAGE;
 [.ZECONT]->ZENEXT;
END;



COMMENT 'CONTINUES WITH THE CURRENT PROCEDURE';

FUNCTION ZECONT;
 IF ZEAGAIN=0 AND ZEJOINT AND .ZENOGAME AND WMESSAGE.ISLINK
 THEN .ZEALERT
 CLOSE;
 1->ZEAGAIN;
 IF .ZENOGAME THEN .ZEROUT ELSE .ZEGAME CLOSE;
END;



COMMENT 'INTERPRETER FOR ROUTINES';

FUNCTION ZEROUT;
 VARS C N;
 IF .ZEATMARK THEN .ZEREVERT EXIT;
 PLACE.ZATAKE->N;
 SUC(NAME.ZATAKE.VALOF,N)->C;
 IF C.HD="*" AND (N.ZAENTRY/=NIL)
 THEN ZAPUT(PLACE,N+1)
 ELSE .ZOBUG; [%".",C.TL.HD%].EVAL
 CLOSE;
END;
```

```
COMMENT 'INTERPRETER FOR GAMES';

FUNCTION ZEGAME;
 VARS C N MINE ENTRY MADE MESSAGE TESTS;
 PLACE.ZATAKE->N;
 SUC(NAME.ZATAKE.VALOF,N)->C;
 IF C=UNDEF THEN .ZEEXIT EXIT;
 (C.HD="*")->ENTRY;
 IF ENTRY THEN (N.ZAENTRY.ISLINK)->MADE CLOSE;
 (C.TL.HD=COLOUR.ZATAKE)->MINE;
 IF MINE
 THEN IF ENTRY
      THEN IF MADE THEN GOTO ADVANCE ELSE GOTO PERFORM CLOSE;
      ELSE GOTO PERFORM
      CLOSE;
 ELSE IF ENTRY
      THEN IF MADE THEN GOTO ADVANCE ELSE GOTO READ CLOSE;
      ELSE GOTO ADVANCE
      CLOSE;
 CLOSE;
PERFORM:
 .ZOBUG; [%".",C.TL.TL.HD%].EVAL; RETURN;
ADVANCE:
 ZAPUT(PLACE,N+1); RETURN;
READ:
 C.TL.TL.TL.TL.REV.TL.REV->TESTS;
 TESTS.ZEREAD->MESSAGE;
 IF MESSAGE.NULL THEN GOTO SWAP
 ELSEIF MESSAGE=[INAPT] THEN GOTO MOAN
 ELSE ZAENTER(MESSAGE)
 EXIT;
MOAN:
 ZEPLAY("JUMP",UNDEF); RETURN;
SWAP:
 [.ZESWAP]->ZENEXT;
END;



COMMENT 'RETURNS 1 IF ROBOT HAS BEEN CALLED BY PARTNER, AND 0 IF NOT';

FUNCTION ZECALLED;
 WMESSAGE=[%ZKME%];
END;




COMMENT 'RESPONDS TO BEING CALLED';

FUNCTION ZEREADY;
 .ZEALERT; [YES].ZEPOST;
END;
```

```
COMMENT 'LOADS ZGGAME AND TAKES BLACK';

FUNCTION ZEALERT;
 "ZGGAME".ZELOAD;
 ZAPUT(COLOUR,BLACK);
END;



COMMENT 'RETURNS 1 IF TOP PROCEDURE IN CONTROL IS NOT A GAME, ELSE 0';

FUNCTION ZENOGAME;
 ZATAKE(KIND)="ROUTINE";
END;



COMMENT 'RETURNS 1 IF PLACE REACHED IS MARKED, AND 0 IF NOT';

FUNCTION ZEATMARK;
 [%NAME.ZATAKE,PLACE.ZATAKE%]=ZEPLACE;
END;



COMMENT 'RESTORES THE INTERRUPTED GAME WHEN A MARK IS REACHED';

FUNCTION ZEREVERT;
 ZEHOLD::ZECONTROL->ZECONTROL;
 NIL->ZEPLACE; NIL->ZEHOLD;
END;



COMMENT 'EXITS FROM THE CURRENT ROUTINE OR GAME';

FUNCTION ZEEXIT;
 IF PRO THEN .ZOSP; 1.SP; NAME.ZATAKE.PR;
 ' ENDED BY '.PRS; ZKME.PR CLOSE;
 ZECONTROL.TL->ZECONTROL;
END;
```

```
COMMENT 'LOADS GAME OR ROUTINE W INTO ZECONTROL';

FUNCTION ZELOAD W;
 IF PRO THEN 2.NL; SP(1+(ZECONTROL.LENGTH*2)); W.PR;
 ' LOADED BY '.PRS; ZKME.PR CLOSE; W.ZAKILL;
 IF W.ZCGAME THEN GOTO XGAME CLOSE;
XROUTINE:
 REP(ZCFSHELL,NAME,W)::ZECONTROL->ZECONTROL;
 ZAPUT(PLACE,1);
 ZAPUT(ENTRIES,W.ZAENTS);
 RETURN;
XGAME:
 REP(ZCGSHELL,NAME,W)::ZECONTROL->ZECONTROL;
 ZAPUT(PLACE,1);
 ZAPUT(COLOUR,WHITE);
 ZAPUT(ENTRIES,W.ZAENTS);
END;



COMMENT 'ARRANGES FOR GAME G TO BE PLAYED WITH FIRST MOVE M, CALLING
         PARTNER IF NECESSARY IN ORDER TO GET ZGGAME LOADED';

FUNCTION ZEPLAY G M;
 [%G%]->ZEGAME1;
 M->ZEMOVE1;
 IF ZEJOINT AND .ZANOGAME AND G/="JUMP"
 THEN "ZGGAME".ZELOAD; [.ZECONT]->ZENEXT
 ELSE [.ZECALL]->ZENEXT;
 CLOSE;
 IF ZEHOLD.ISLINK AND SUC(ZEHOLD,NAME)="ZGGAME"
 THEN NIL->ZEHOLD; NIL->ZEPLACE
 CLOSE;
END;



COMMENT 'POSTS MESSAGE M';

FUNCTION ZEPOST M;
 M->ZEBOX;
 [.ZESEND]->ZENEXT;
END;
```

```
COMMENT 'TRANSLATES MESSAGE FROM ENGLISH INTO AN ENTRY, USING THE
         LIST OF FUNCTIONS T OBTAINED FROM THE GAME DEFINITION';

FUNCTION ZEREAD T;
 VARS M E;
 WMESSAGE->M;
 NIL->WMESSAGE;
 IF M.NULL THEN NIL EXIT;
LOOP:
 IF T.NULL THEN [INAPT] EXIT;
 [%M,".",T.HD%].EVAL->E; T.TL->T;
 IF E=UNDEF THEN GOTO LOOP ELSE E CLOSE;
END;



COMMENT 'GIVES CONTROL TO STRUCTURE BELOW CURRENT ONE AND
         MARKS IT AT PLACE N';

FUNCTION ZEMARK P;
 P->ZEPLACE; ZECONTROL.HD->ZEHOLD; ZECONTROL.TL->ZECONTROL;
END;



COMMENT 'USED TO INITIALISE THE ZE VARIABLES BEFORE A RUN';

FUNCTION ZEPREP;
 NIL->ZECONTROL; "ZRBASIC".ZELOAD; 0->ZEJOINT;
 [.ZECONT]->ZENEXT; NIL->ZEPLACE; 1->ZEAGAIN;
END;



COMMENT 'JUMPS BACK TO ZRBASIC 7. USED WHEN AN INAPPROPRIATE
         REMARK HAS BEEN MADE';

FUNCTION ZEJUMP;
 "ZRBASIC".ZAJUMP; ZAGOTO(7);
END;
```

```
*** [WRITE] ***

COMMENT 'THESE FUNCTIONS TRANSLATE GAME ENTRIES INTO
         ENGLISH IN AN UNPRINCIPLED MANNER';



COMMENT 'ZGGAME 1 AND 2';

FUNCTION ZWNAME X;
[ZGASK [MAY I ASK YOU SOMETHING]
 ZGTELL [I WANT TO TELL YOU SOMETHING]
 ZGRULE [I WANT TO EXPLAIN SOMETHING]
 ZGGOAL [I WANT TO SUGGEST A GOAL]
 ZGASSESS [LETS ASSESS THE RESULT OF MY ACTION]
 ZGPLAN [SHALL WE MAKE A PLAN]
 JUMP [WE HAVE GOT MUDDLED; LETS START AGAIN]
],X.HD.SUC;
END;



FUNCTION ZWREADY X;
 IF MEMB(1.ZAENTRY.HD,[ZGPLAN ZGCHECK ZGASSESS JUMP])
 THEN [OK] ELSE [GO AHEAD] CLOSE;
END;



COMMENT 'ZGASK 1 AND 2';

FUNCTION ZWQUERY X;
 VARS V G; X.HD->V; X.TL->G;
 IF G.HD=ZKME THEN [I]<>G.TL->G;IF V="IS" THEN "AM"->V CLOSE;CLOSE;
 IF G.HD=ZKYOU THEN [YOU]<>G.TL->G;
 IF V="IS" THEN "ARE"->V CLOSE;
 CLOSE;
 IF MEMB(G.HD,[DOOR BOLT]) THEN [THE]<>G->G CLOSE;
 IF G.TL.HD=PUSH THEN G<>[THE DOOR]->G
 ELSEIF G.TL.HD=SLIDE THEN G<>[THE BOLT]->G CLOSE;
 V::G;
END;



FUNCTION ZWANSWER X;
 IF X=[UNDEF] THEN [I DONT KNOW] ELSE X CLOSE;
END;
```

```
COMMENT 'ZGGOAL 1 AND 2';

FUNCTION ZWPLEAD X;
 IF X.HD=ZKME THEN X.TL
 ELSEIF X.HD=ZKYOU THEN [YOU]<>X.TL
 ELSE [THE]<>X
 CLOSE; ->X;
 [WILL YOU HELP ME GET]<>X;
END;



FUNCTION ZWREACT X;
 IF X=[YES] THEN [BY ALL MEANS] ELSE [NO] CLOSE;
END;



COMMENT 'ZGTELL 1 AND 2';

FUNCTION ZWRELATE X;
 IF X.ZCEVENT
 THEN [PUSH [I HAVE PUSHED THE DOOR]
       SLIDE [I HAVE SLID THE BOLT]
       MOVE [I HAVE MOVED]],X.TL.HD.SUC
 EXIT;
 VARS T; X.HD->T; X.TL.ZWQUERY->X;
 IF X.TL.HD="THE"
 THEN [THE]<>[%X.TL.TL.HD,X.HD%]; X.TL.TL.TL->X
 ELSE [%X.TL.HD,X.HD%]; X.TL.TL->X;
 CLOSE; IF T=0 THEN <>[NOT] CLOSE; <>X;
END;



FUNCTION ZWEXAMINE X;
 X.ZWWREPLY;
END;



COMMENT 'ZGRULE 1,2,4 AND 5';

FUNCTION ZWWRULE X;
 VARS E S R;
 SUC(X,EVT)->E; SUC(X,SIT)->S; SUC(X,RES)->R;
 [IF]<>TL(ZWQUERY("CAN"::(ZKYOU::E.TL)))->E;
 IF S=[ANY]
 THEN NIL
 ELSEIF S.HD/="ROBOT" THEN [%"WHEN","THE",S.HD,"IS",S.TL.HD%]
 ELSE [WHEN YOU ARE]<>[%S.TL.HD%]
 CLOSE; ->S;
 IF R=[NOTHING] THEN [, NOTHING HAPPENS]
 ELSE IF R.HD="ROBOT" THEN [, YOU CHANGE] ELSE [, THE]<>R
 <>[CHANGES] CLOSE; <>[POSITION] CLOSE; ->R; E<>S<>R;
END:
```

```
FUNCTION ZWWREPLY X;
[YES [I ALREADY KNOW THAT]
 NO [I DISAGREE]
 UNDEF [I SEE]
],X.HD.SUC;
END;



FUNCTION ZWBRULE X;
 X.ZWWRULE;
END;



FUNCTION ZWBREPLY X;
 X.ZWWREPLY;
END;



COMMENT  'ZGPLAN 1 AND 2';

FUNCTION ZWSUGGEST X;
 VARS S;
 IF X=[NO] THEN [I CANT THINK OF ONE] EXIT;
 X.ZPSIT->S;
 IF S/=UNDEF
 THEN TL(ZWQUERY([IS]<>S))->S;IF S.HD="I" THEN "ME"::S.TL->S CLOSE;
      [WE GET]<>S<>[AND THEN]
 ELSE NIL
 CLOSE; ->S; X.ZPEVT->X;
 TL(ZWQUERY([CAN]<>X))->X;
 [I SUGGEST THAT]<>S<>X;
END;



FUNCTION ZWRESPOND X;
 VARS P; 1.ZAENTRY->P;
 IF P=[NO]
 THEN [YES [I WILL THEN] NO [I CANT EITHER]]
 ELSE [YES [ALL RIGHT] NO [I DISAGREE]]
 CLOSE; X.HD.SUC;
END;
```

```
COMMENT 'ZGASSESS 1,2,3 AND 4';

FUNCTION ZWCHANGE X;
 IF X=[NOTHING] THEN [NOTHING HAS HAPPENED]; RETURN
 ELSEIF X.HD=ZKME THEN [I HAVE]
 ELSEIF X.HD=ZKYOU THEN [YOU HAVE]
 ELSE [THE]<>X<>[HAS]
 CLOSE; <>[CHANGED POSITION];
END;



FUNCTION ZWCONFIRM X;
 [YES [YES] NO [I DISAGREE]],X.HD.SUC;
END;



FUNCTION ZWPARENT X;
 X.ZWRELATE.REV->X; X.HD;
 IF MEMB("NOT",X) THEN ::[YET] ELSE ::[NOW] CLOSE;
 <>X.TL->X; X.REV;
END;



FUNCTION ZWBENTER X;
 [RIGHT];
END;
```

```
*** [DECIPHER] ***

COMMENT 'USED TO DECODE ENGLISH UTTERANCES';



COMMENT 'USED MAINLY TO REPLACE VARIANT FORMS OF A WORD (E.G. AM,
         ARE) WITH A STANDARD ONE (IS)';

FUNCTION ZDCLEAN X;
 VARS L1 L2; NIL->L1;
 X,"SLID",SLIDE.XCH; "PUSHED",PUSH.XCH; "MOVED",MOVE.XCH->X;
 X,"AM","IS".XCH; "ARE","IS".XCH->X;
 IF MEMB("HELP",X) AND X.LENGTH>4 THEN X.TL.TL.TL.TL->X CLOSE;
 IF X.LENGTH>1 AND MEMB(X.TL.HD,[SUGGEST WANT]) THEN X.TL.TL->X CLOSE;
 IF MEMB(SLIDE,X) THEN ZDSPLIT(X,SLIDE)->L1->L2; L1<>[SLIDE]->L1
 ELSEIF MEMB(PUSH,X) THEN ZDSPLIT(X,PUSH)->L1->L2; L1<>[PUSH]->L1
 CLOSE;
 IF L1.ISLINK AND L2.LENGTH>1 THEN L1<>L2.TL.TL->X CLOSE;
 X;
END;



COMMENT 'REPLACES PRONOUNS WITH THEIR REFERENTS';

FUNCTION ZDPRONS L;
 L,"I",ZKYOU.XCH; "ME",ZKYOU.XCH;"YOU",ZKME.XCH;"WE",BOTH.XCH;
END;



COMMENT 'GIVEN A LIST OF WORDS X AND AN ENGLISH EXPRESSION L,
         RETURNS THE FIRST WORD IN X WHICH IS ALSO IN L, AND
         UNDEF IF NONE ARE';

FUNCTION ZDFIND L X;
 LOOPIF X.ISLINK
 THEN IF MEMB(X.HD,L) THEN X.HD; RETURN ELSE X.TL->X CLOSE;
 CLOSE; UNDEF;
END;



COMMENT 'RETURNS A LIST OF ALL THE WORDS BEYOND X IN L';

FUNCTION ZDHALF L X;
 LOOPIF L.HD/=X THEN L.TL->L CLOSE; L.TL;
END;
```

```
COMMENT 'RETURNS LISTS OF THE WORDS BEFORE AND AFTER X IN L';

FUNCTION ZDSPLIT L X;
 ZDHALF(L,X); REV(ZDHALF(L.REV,X));
END;



COMMENT 'THE FOLLOWING FUNCTIONS ARE USED TO DECODE DIFFERENT
         KINDS OF UTTERANCE. IF THE ENGLISH EXPRESSION X CAN
         BE CONSTRUED AS THE KIND OF ENTRY WANTED, THE ENTRY
         IS RETURNED; IF NOT, UNDEF IS RETURNED. THUS ZDGAME
         TRIES TO INTERPRET X AS A GAME SUGGESTION, ZDQUERY
         TRIES TO INTERPRET IT AS A QUERY, AND SO ON';



FUNCTION ZDGAME X;
 VARS L;
 [ASK ZGASK GOAL ZGGOAL NEXT ZGCHECK EXPLAIN ZGRULE
 TELL ZGTELL ASSESS ZGASSESS PLAN ZGPLAN START JUMP]->L;
 SUC(L,ZDFIND(L,X))->L;
 IF L=UNDEF THEN UNDEF ELSE L::NIL CLOSE;
END;



FUNCTION ZDQUERY X;
 VARS A;
 X.ZDCLEAN.ZDPRONS->X;
 IF X.LENGTH<3 OR NOT(MEMB(X.HD,[CAN IS])) THEN UNDEF EXIT;
 IF X.HD="CAN" THEN X.TL.ZDEVENT ELSE X.TL.ZDSITN CLOSE; ->A;
 IF A=UNDEF THEN UNDEF ELSE X.HD::A CLOSE;
END;



FUNCTION ZDEVENT X;
 X.ZDCLEAN.ZDPRONS->X;
 IF X.LENGTH<2 THEN UNDEF EXIT;
 (ZDFIND([JOHN MARY],X))::[%ZDFIND(ZCACTS,X)%]->X;
 IF MEMB(UNDEF,X) THEN UNDEF ELSE X CLOSE;
END;



FUNCTION ZDSITN X;
 X.ZDCLEAN.ZDPRONS->X; X<>[%ZKYOU%]->X;
 IF X.LENGTH<2 THEN UNDEF EXIT;
 (ZDFIND(ZCOBJS,X))::[%ZDFIND(ZCPROPS,X)%]->X;
 IF MEMB(UNDEF,X) OR NOT(ZCPROPOF(X.TL.HD,X.HD.ZCTYPOF))
 THEN UNDEF
 ELSE X
 CLOSE;
END;
```

```
FUNCTION ZDRULE X;
 VARS E S R;
 XCH(X.ZDCLEAN,"YOU","ROBOT")->X;
 IF X.HD/="IF" OR NOT(MEMB(",",X)) THEN UNDEF EXIT;
 ZDSPLIT(X,",")->E->R;
 IF MEMB("WHEN",E) THEN ZDSPLIT(E,"WHEN")->E->S ELSE [ANY]->S CLOSE;
 ZDFIND(ZCACTS,E)->X;
 IF MEMB("ROBOT",E) AND X/=UNDEF
 THEN [ROBOT]<>[%X%]->E
 ELSE UNDEF
 EXIT;
 IF S/=[ANY]
 THEN (ZDFIND(ZCTYPES,S))::[%ZDFIND(ZCPROPS,S)%]->S;
      IF MEMB(UNDEF,S) OR NOT(ZCPROPOF(S.TL.HD,S.HD))
      THEN UNDEF
      EXIT;
 CLOSE;
 ZDFIND("NOTHING"::ZCTYPES,R)::NIL->R; IF R=UNDEF THEN UNDEF EXIT;
 [%EVT,E,SIT,S,RES,R%];
END;



FUNCTION ZDPLAN X;
 VARS S E;
 X.ZDCLEAN.ZDPRONS->X;
 IF MEMB("AND",X) THEN ZDSPLIT(X,"AND")->S->E
 ELSEIF MEMB("THEN",X) THEN ZDSPLIT(X,"THEN")->S->E
 ELSE NIL->S; X->E
 CLOSE;
 IF S.ISLINK THEN S.ZDSITN->S CLOSE;
 E.ZDEVENT->E;
 IF S=UNDEF OR E=UNDEF THEN UNDEF EXIT;
 IF S.ISLINK THEN BOTH::[%S%]->S CLOSE;
 S<>[%E.HD,E%]
END;



FUNCTION ZDSIGN X;
 VARS Y N;
 IF X=[I SEE] THEN [UNDEF] EXIT;
 IF MEMB("DONT",X) AND MEMB("KNOW",X) THEN [UNDEF] EXIT;
 [KNOW YES AGREE GOOD CAN WILL RIGHT FINE SPLENDID OK GO BY]->Y;
 [NO DISAGREE CANT WONT BAD LOUSY FAULTY SILLY]->N;
 ZDFIND(N<>Y,X)->X;
 IF X=UNDEF THEN UNDEF ELSEIF MEMB(X,Y) THEN [YES] ELSE [NO] CLOSE;
END;
```

```
FUNCTION ZDFACT X;
 VARS T; X.ZDCLEAN->X;
 MEMB("NOT",X).NOT->T;
 (ZDFIND(X,[CAN IS])::X).ZDQUERY->X;
 IF X=UNDEF THEN UNDEF ELSE T::X CLOSE;
END;



FUNCTION ZDOBJ X;
 ZDFIND([NOTHING]<>ZCOBJS,X.ZDPRONS); ::NIL;
END;
```

```
*** [PLAY] ***

COMMENT 'THE FUNCTIONS USED TO SET A RUN OF THE PROGRAM GOING';


VARS COUNT STOP PRO PRW1; 1->COUNT; 150->STOP; 0->PRW1; 0->PRO;


COMMENT 'THE FIRST FEW FUNCTIONS ARE USED TO START A CONVERSATION
         BETWEEN MARY AND A HUMAN OPERATOR';


COMMENT 'STARTS THE RUN';

FUNCTION GO;
 .SET; ZAENTER1("ZRBASIC",4,[FAILED]);1->COUNT;.PLAY;
END;


COMMENT 'INITIALISES THE CRITICAL VARIABLES';

FUNCTION SET;
 JOHN->ZKYOU;  MARY->ZKME;
 [DOOR OPEN]->ZKGOAL; NIL->WMESSAGE;
 [MARY IN JOHN OUT DOOR SHUT BOLT DOWN]->WOBJECTS;
 [SLIDE PUSH MOVE]->ZKACTS; [JOHN MARY BOLT]->ZKSEE; .ZEPREP;
 [[EVT[ROBOT PUSH]SIT[ANY]RES[NOTHING]]
  [EVT[ROBOT MOVE]SIT[ANY]RES[NOTHING]]
  [EVT[ROBOT SLIDE]SIT[ANY]RES[NOTHING]]]->ZCRSHELL;
END;


COMMENT 'THE CHAIRMAN';

FUNCTION PLAY;
 VARS X; COUNT//2; .ERASE; ->X; 3.NL; .PRW; 1.NL;
LOOP:
 IF COUNT>STOP THEN 3.NL; EXIT;
 IF X THEN .ZEAROUSE; IF PRO THEN 1.NL CLOSE;
 2.NL; COUNT.PR; ' MARY:'.PRS; WMESSAGE.WRITE; COUNT+1->COUNT
 ELSE 1->X CLOSE; IF COUNT>STOP THEN 3.NL EXIT;
 3.NL; COUNT.PR; ' JOHN'.PRS; .READ->WMESSAGE; COUNT+1->COUNT;
 GOTO LOOP;
END;
```

```
COMMENT 'PRINTS OUT A ROBOTS UTTERANCES';

FUNCTION WRITE L;
 IF L.NULL THEN ' --'.PRS EXIT;
 LOOPIF L.ISLINK
 THEN IF L.HD="SOMETHIN" THEN ' SOMETHING'.PRS
 ELSEIF L.HD="," OR L.HD=";" THEN L.HD.PR
 ELSE 1.SP; L.HD.PR CLOSE; L.TL->L;
 CLOSE;
 '.'.PRS;
END;



COMMENT 'READS IN THE OPERATORS UTTERANCES';

FUNCTION READ;
 VARS X L; NIL->L;
LOOP: .ITEMREAD->X; IF X="--" THEN NIL EXIT;
 IF X="." THEN L.REV ELSE X::L->L; GOTO LOOP CLOSE;
END;



COMMENT 'THE REST OF THE FUNCTIONS ARE USED FOR A CONVERSATION
         BETWEEN JOHN AND MARY';



COMMENT 'ONE WAY OF PRE-SETTING THE CRITICAL VARIABLES';

FUNCTION RESET1;
 JOHN,JOHN->JKME->MKYOU;
 MARY,MARY->MKME->JKYOU;
 [JOHN IN]->JKGOAL; [NONE]->MKGOAL;
 [JOHN OUT MARY IN BOLT UP DOOR SHUT]->WOBJECTS;
 NIL->WMESSAGE;
 [SLIDE MOVE]->JKACTS; [PUSH MOVE]->MKACTS;
 [[EVT[ROBOT PUSH]SIT[BOLT UP]RES[DOOR]]
  [EVT[ROBOT SLIDE]SIT[ROBOT IN]RES[BOLT]]
  [EVT[ROBOT MOVE]SIT[ANY]RES[NOTHING]]]->JCRSHELL;
 [[EVT[ROBOT PUSH]SIT[ANY]RES[NOTHING]]
  [EVT[ROBOT SLIDE]SIT[ANY]RES[NOTHING]]
  [EVT[ROBOT MOVE]SIT[DOOR OPEN]RES[ROBOT]]]->MCRSHELL;
 [DOOR JOHN BOLT MARY]->JKSEE; [MARY]->MKSEE;
 .JEPREP; .MEPREP;
END;
```

```
COMMENT 'THE CHAIRMAN`;

FUNCTION REPLAY;
 VARS X Y; COUNT//2; .ERASE; ->X; 0->Y;
 IF X THEN GOTO JOHN ELSE GOTO MARY CLOSE;
JOHN:
 IF COUNT>STOP THEN 3.NL EXIT; .JEAROUSE;
 IF WMESSAGE.NULL
 THEN IF Y THEN 3.NL EXIT;
      IF PRO THEN 3.NL; '** JOHN SWAPS`.PRS; 1.NL CLOSE;
      1->Y; GOTO MARY
 CLOSE;
 IF Y=0 OR PRO OR PRW1
 THEN 3.NL; IF PRO THEN '**`.PRS CLOSE;
      IF Y THEN COUNT-1->COUNT CLOSE; COUNT.PR;
      1+COUNT->COUNT; ' JOHN:`.PRS
 CLOSE; 0->Y; 0->PRW1;
 WMESSAGE.WRITE; IF PRO THEN 1.NL CLOSE;
MARY:
 IF COUNT>STOP THEN 3.NL EXIT; .MEAROUSE;
 IF WMESSAGE.NULL
 THEN IF Y THEN 3.NL EXIT;
      IF PRO THEN 3.NL; '** MARY SWAPS`.PRS; 1.NL CLOSE;
      1->Y; GOTO JOHN
 CLOSE;
 IF Y=0 OR PRO OR PRW1
 THEN 3.NL; IF PRO THEN '**`.PRS CLOSE;
      IF Y THEN COUNT-1->COUNT CLOSE; COUNT.PR;
      1+COUNT->COUNT; ' MARY:`.PRS
 CLOSE; 0->Y; 0->PRW1;
 WMESSAGE.WRITE; IF PRO THEN 1.NL CLOSE; GOTO JOHN;
END;



COMMENT 'RUNS A CONVERSATION WITH THE FIRST INITIAL SETTING`;

FUNCTION RUN1;
 .RESET1; 1->COUNT; JAENTER1("JRBASIC",4,[FAILED]);
 3.NL; .PRW; .REPLAY;
END;
```

```
COMMENT 'ANOTHER WAY OF SETTING THE VARIABLES';

FUNCTION RESET2;
 JOHN,JOHN->JKME->MKYOU;
 MARY,MARY->MKME->JKYOU;
 [DOOR SHUT]->JKGOAL; [NONE]->MKGOAL;
 [JOHN IN MARY OUT BOLT DOWN DOOR OPEN]->WOBJECTS;
 NIL->WMESSAGE;
 [PUSH]->JKACTS; [MOVE SLIDE]->MKACTS;
 NIL->JKSEE; [JOHN MARY BOLT DOOR]->MKSEE;
 [[EVT[ROBOT PUSH]SIT[ANY]RES[NOTHING]]
  [EVT[ROBOT SLIDE]SIT[ANY]RES[NOTHING]]
  [EVT[ROBOT MOVE]SIT[ANY]RES[NOTHING]]]->JCRSHELL;
 [[EVT[ROBOT PUSH]SIT[ANY]RES[DOOR]]
  [EVT[ROBOT SLIDE]SIT[ANY]RES[BOLT]]
  [EVT[ROBOT MOVE]SIT[ANY]RES[ROBOT]]]->MCRSHELL;
 .JEPREP; .MEPREP;
END;


COMMENT 'RUNS THE CONVERSATION WITH THE SECOND SETTING';

FUNCTION RUN2;
 .RESET2; 1->COUNT;JAENTER1("JRBASIC",4,[FAILED]); 3.NL; .PRW; .REPLAY;
END;
```

```
*** [PRINT] ***

COMMENT 'VARIOUS FUNCTIONS USED TO PRINT OUT THE STATE OF
         THE PROGRAM (E.G. ZOKNOW PRINTS OUT A ROBOTS WORLD MODEL)';



FUNCTION ZOPLEAD X;
 X.ZWPLEAD.TL.TL->X;
 IF X.LENGTH=3 THEN [%X.HD,X.TL.HD,ZKME,X.TL.TL.HD%] ELSE X CLOSE;
END;



FUNCTION ZOBUG;
 IF PRO THEN .ZOSP; PLACE.ZATAKE.PR; ' CALLED BY '.PRS; ZKME.PR CLOSE;
END;



FUNCTION ZOSP;
 2.NL; SP(LENGTH(ZECONTROL.TL)*2);
END;



FUNCTION ZOYN X;
 IF X=1 THEN "YES" ELSEIF X=0 THEN "NO" ELSE UNDEF CLOSE;
END;



FUNCTION ZOKNOW;
 VARS X Y;
 2.NL; '*** WHAT '.PRS; ZKME.PR; ' KNOWS ***'.PRS;
 2.NL; 'A. WORLD'.PRS; 2.NL; '    1. POSITION OF OBJECTS'.PRS;
 1.NL; ZKWORLD->X;
 LOOPIF X.ISLINK THEN 1.NL; 3.SP; X.HD.PR; ': '.PRS; X.TL.HD.PR;
 X.TL.TL->X;
 CLOSE;
 2.NL; '    2. CONSEQUENCES OF EVENTS'.PRS; 1.NL; ZKRULES->X;
 LOOPIF X.ISLINK
 THEN 1.NL; 3.SP; SUC(X.HD,EVT).TL.HD.PR; ':'.PRS; 1.NL; 2.SP;
 IF SUC(X.HD,RES)=[UNDEF] THEN ' UNDEF'.PRS ELSE X.HD.ZWWRULE.WRITE
 CLOSE; X.TL->X;
 CLOSE;
 2.NL; 'B. '.PRS; ZKYOU.PR; 2.NL; '    1. GOAL:'.PRS;
 IF ZKXGOAL.NULL THEN ' UNDEF'.PRS ELSE ZKXGOAL.ZOPLEAD.WRITE
 CLOSE; 2.NL; '    2. RANGE OF ACTIONS'.PRS; 1.NL; ZCACTS->X;
 LOOPIF X.ISLINK THEN 1.NL; 3.SP; X.HD.PR; ': '.PRS;
 SUC(ZKXACTS,X.HD).ZOYN.PR; X.TL->X;
 CLOSE;
```

```
2.NL; '   3. KNOWLEDGE OF OBJECT POSITIONS'.PRS; 1.NL;
ZKXSEE->X;
LOOPIF X.ISLINK THEN 1.NL; 3.SP; X.HD.PR; ': '.PRS;
X.TL.HD.ZOYN.PR; X.TL.TL->X;
CLOSE;
2.NL; '   4. BELIEFS ABOUT CONSEQUENCES OF EVENTS'.PRS;
1.NL; ZKXRULES->X; NIL->Y;
LOOPIF X.ISLINK
THEN 1.NL; 3.SP; SUC(X.HD,EVT).TL.HD::Y->Y; Y.HD.PR; ':'.PRS;
1.NL; 2.SP; X.HD.ZWWRULE.WRITE; X.TL->X;
CLOSE; ZCACTS->X;
LOOPIF X.ISLINK
THEN IF MEMB(X.HD,Y) THEN ELSE 1.NL; 3.SP; X.HD.PR; ':'.PRS; 1.NL;
3.SP; UNDEF.PR; CLOSE; X.TL->X;
CLOSE; 2.NL; '**********************'.PRS; 2.NL;
END;



FUNCTION ZOPLAN;
 2.NL; '*** '.PRS; ZKME.PR; 'S PLAN ***'.PRS;
 2.NL; 'GOALS INVOLVED'.PRS; ZPGOAL.PRA;
 2.NL; 'SUBGOALS OF EACH GOAL'.PRS; ZPTREE.PRA;
 2.NL; 'ACTORS RESPONSIBLE'.PRS; ZPACTOR.PRA;
 2.NL; 'STATE OF EACH GOAL'.PRS; ZPSTATE.PRA;
 2.NL; '**********************'.PRS; 2.NL;
END;



FUNCTION ZOCONT;
 VARS X; 2.NL; '***  '.PRS; ZKME.PR; 'S CONTROL STRUCTURE ***'.PRS;
 ZECONTROL.REV->X;
 LOOPIF X.ISLINK THEN X.HD.PRC; X.TL->X; CLOSE;
 '**********************'.PRS; 2.NL;
END;



FUNCTION ZOMIND;
   4.NL; '*** CURRENT STATE OF '.PRS; ZKME.PR; 'S MIND ***'.PRS;
  .ZOKNOW; .ZOPLAN; .ZOCONT;
END;
```

References

ALSTON, W. (1964) "Philosophy of Language",
Englewood Cliffs, N.J. Prentice Hall.

AUSTIN, J. (1962) "How to do things with words",
Cambridge, Mass., H.U.P.

BOBROW, D. (1964) "Natural Language Input for a Computer Problem-Solving System",
Unpublished Doctoral thesis, M.I.T., Cambridge, Mass.

BURSTALL, POPPLESTONE and COLLINS (1972)
"Programming in POP-2", E.U.P., Edinburgh.

CARBONELL, J. (1970) "Mixed Initiative Man-Computer Instructional Dialogues",
Doctoral Dissertation, M.I.T., Cambridge, Mass.

CARBONELL and COLLINS (1973) "Natural Semantics in Artificial Intelligence",
Bolt, Beranek and Newman Inc., Cambridge, M.I.T.

CHOMSKY, N. (1957) "Syntactic Structures",
The Hague, Mouton.

CHOMSKY, N. (1965) "Aspects of the Theory of Syntax",
Cambridge, M.I.T.

COLBY, WEBER and HILF (1971) "Artificial Paranoia",
Artificial Intelligence, 1971, volume II.

COLLINS, CARBONELL and WARNOCK (1972) "Semantic Information Processing by Computer",
Proceedings of the International Congress of Cybernetics and Systems, Oxford, England.

DREYFUS, H. (1972) "What Computers Can't Do",
Harper and Row, New York.

FOX, L. (Ed.) (1966) "Advances in Programming and Non-Numerical Computation",
Pergamon, Oxford.

FEIGENBAUM and FELDMAN (Eds.) (1963) "Computers and Thought",
McGraw-Hill, New York.

HEWITT, C. (1971) "Description and Theoretical Analysis (Using Schemes) of PLANNER",
Unpublished Doctoral Dissertation, M.I.T., Cambridge, Mass.

LYONS, J. (1970) "Chomsky",
Fontana, London.

MINSKY, M. (1968) "Semantic Information Processing",
M.I.T. Press, Mass.

MINSKY and PAPERT (1972) "Artificial Intelligence Progress Report",
Artificial Intelligence Memo No. 252, M.I.T., Cambridge, Mass.

QUILLIAN, R. (1967) "Word Concepts: A Theory and a Simulation of Some Basic Semantic Capabilities",
Behavioural Science 12, 1967.

SEARLE, J. (1971) "Speech Acts",
Cambridge, C.U.P.

WEIZENBAUM, J. (1966) "ELIZA - a Program for the Study of Natural Language Communication between Man and Machine",
Comm. ACM 9, 1966.

WINOGRAD, T. (1972) "Understanding Natural Language",
E.U.P., Edinburgh.